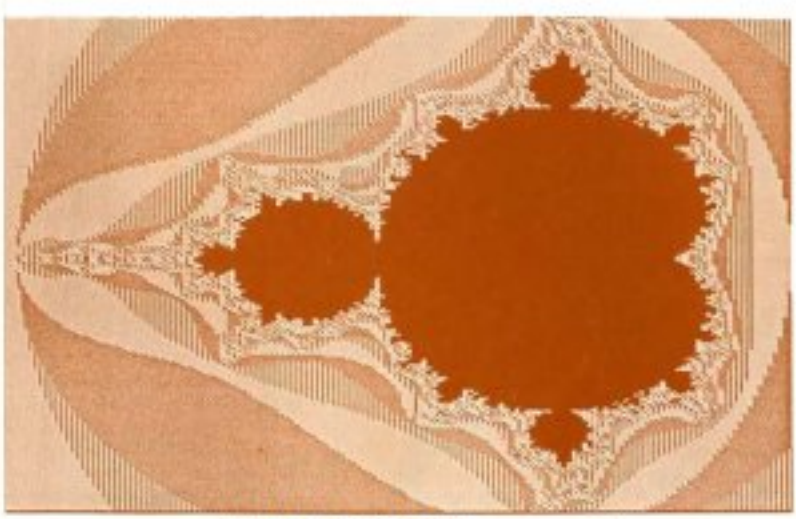
COMAL TODAY 18



C64 MOUSE (page 16)



The Doc Box™ ... and CP/M COMAL Arrive. See page 77



WHAT IS IT? (see page 53)

BLACKJACK
page 55

KNIGHTS TOUR
page 56

PRINT SHOP CONVERTER
page 46

ZIP ZONE
page 40

FRACTAL GEOMETRY
page 53

REVEAL PROCS
page 60

THIRD DIMENSION
page 52

REVERSI
page 24

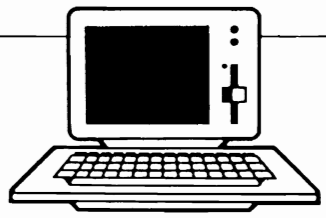
MULTIPLE PROGRAMS
page 49

PUZZLE
page 36

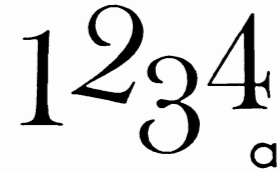
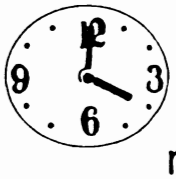
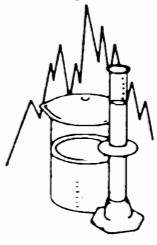
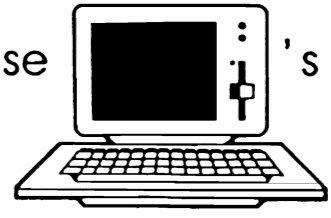
COMAL Today
6041 Monona Drive
Madison, WI 53716

Bulk Rate
U.S.Postage
Paid
Madison, WI
Permit 2081

If your label says
Last Issue: 18
You must renew now.
Use order form inside.

If you use a  **, you need**

The Computing Teacher

We publish articles from all over the  in **The Computing Teacher** journal so that you'll get the best information. Information that's crystal clear, interesting  and fun to use in the classroom. You can  on us to have accurate, timely articles and to save you  and  with our  reviews and new product releases. We have columns for  , , Logo, language arts and computers in the library. **The Computing Teacher**—for all those who use  's in the classroom.

ICCE, U of O, 1787 Agate Street, Eugene, OR 97403 USA

COMAL Today Issue 18 September 22, 1987

GENERAL

2 - Editor's Disk - Len Lindsay
3 - Specials (also on page 77)
3 - Vote
4 - Guest Editorial - Robert Ross
5 - Rumors - Buzz (also on page 74)
8 - Questions & Answers
9 - Letters
10 - COMAL Clinic
12 - Bug Fixes
14 - CP/M COMAL
61 - Extra Programs

BEGINNERS

6 - How To ...
17 - A Matter of Style - Bill Inhelder / Staff
33 - First COMAL 0.14 Program - Gary Franklin

FUN and GRAPHICS

13 - Cryptograms - Bill Inhelder / Bob McCauley
24 - Reversi - Captain COMAL
28 - Poetry - Dick Klingens
30 - Hammurabi - Mark Skopinsky
36 - Puzzle - Christopher Laprise
39 - Epicycloids - Jack Baldrige
44 - COMAL Flex - Bill Nissley
52 - Third Dimension - Paul Van Leeuwen
53 - Fractal Geometry - Ted Groszkiewicz
55 - Blackjack - Jeffrey Sherwood
56 - Knights Tour - Bill Inhelder

2.0 PACKAGES & PROGRAMMING

16 - Mouse - Garrett Hughes
44 - MT80 Dump - Bernie Flinn
49 - Multiple Programs - Richard Bain
59 - Edit Source - Dick Klingens
60 - Reveal Procs - Dick Klingens
63 - Super Chip - Joe Potter
64 - De-Link - Dick Klingens
65 - Stacks - Richard Bain

APPLICATIONS

35 - Clock - Gerard Busker
40 - Zip Zone - Len Lindsay
46 - Print Shop - Terry Mills / Craig Hardy
61 - Typing Tutor - Henry Farkas
62 - Spread Sheet - Robert E Webb

REFERENCE

23 - Best Books
71 - X/PL0 - Dick Klingens
73 - Disk Sleeve Directories

ADVERTISERS

ifc - Computing Teacher
27 - QLink
72 - Apple COMAL
75 - Classified Ads
ibc - Transactor

Publisher
COMAL Users Group, U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

Editor
Len Lindsay

Assistants
Richard Bain
Maria Lindsay
David Stidolph
Geoffrey Turney

Artwork
Rhianon Lindsay

Contributors
Morgan Allsup
Richard Bain
Jack Baldrige
B K Ball
Gerard Busker
Captain COMAL
Mark M Disko
Henry Farkas
Bernie Flinn
Gary Franklin
Ted Groszkiewicz
Craig Hardy
Bob Hoerter
Garrett Hughes
Bill Inhelder
Alan Jones

Contributors
Dick Klingens
Christopher Laprise
Len Lindsay
Bob McCauley
Terry R Mills
Bill Nissley
Joe Potter
Joel Rea
Robert Ross
GlennS
Sid Seiferlein
Jeffrey Sherwood
Mark Skopinsky
David Stidolph
Paul Van Leeuwen
Robert E Webb

COMAL Today welcomes contributions of articles, manuscripts and programs which would be of interest to readers. All manuscripts and articles sent to COMAL Today will be treated as unconditionally assigned for publication and copyright purposes to COMAL Users Group, U.S.A., Limited and is subject to the Editor's unrestricted right to edit and to comment editorially. Programs developed and submitted by authors remain their property, with the exception that COMAL Users Group, U.S.A., Limited reserves the right to reprint the materials, based on that published in COMAL Today, in future publications. There will be no remuneration for any contributed manuscripts, articles or programs. These terms may be varied only upon the prior written agreement of the Editor and COMAL Users Group, U.S.A., Limited. Interested authors should contact the Editor for further information. All articles and programs should be sent to COMAL Users Group, U.S.A., Limited, 6041 Monona Drive, Madison, WI 53716. Authors of articles, manuscripts and programs warrant that all materials submitted are original materials with full ownership rights resident in said authors. No portion of this magazine may be reproduced in any form without written permission from the publisher. Local Users Groups may reprint material from this issue if credit is given to COMAL Today and the author. Entire contents copyright (c) 1987 COMAL Users Group, U.S.A., Limited. The opinions expressed in contributed articles are not necessarily those of COMAL Users Group, U.S.A., Limited. Although accuracy is a major objective, COMAL Users Group, U.S.A., Limited cannot assume liability for article or program errors.

Please note these trademarks: Commodore 64, CBM of Commodore Electronics Ltd; PET, Easy Script, Amiga of Commodore Business Machines, Inc; Calvin the COMAL Turtle, Captain COMAL, Super Chip, COMAL Today, Doc Box, Common COMAL of COMAL Users Group, U.S.A., Limited; Buscard, PaperClip of Batteries Included; CP/M of Digital Research; Z-80 of Zilog; IBM of International Business Machines; Apple, MacIntosh of Apple Computer Inc; QLink, Quantum Link of Quantum Computer Service; Compute!, Compute!'s Gazette, Speedscript of Compute! Publications, Inc.; Word Perfect of Word Perfect Corp; UniComal of UniComal; Mytech of Mytech; Atari of Atari; PrintShop of Broderbund; Print Master of Unison World. Sorry if we missed any others.

Editor's Disk



C64 0.14 C64 2.0 IBM CP/M

by Len Lindsay

Finally. A new 24 page COMAL INFO booklet. Now it covers the different implementations and includes a chart of common keywords and their syntax. A copy is included with each package we ship. But if you want one sooner, send us a 39 cent stamped, self addressed envelope.

We have a new booting system for COMAL 0.14. Why? See page 12. We also have added several help files on *Today Disk #18*. If you don't have a recent COMAL 0.14 disk, you may want to get this one!

We are trying something new on *Today Disk #18*. Usually we **SAVE** programs on the disk. You then **LOAD** them. A few readers have requested that some of the programs be LISTed to disk. So, three programs on the front of the disk (0.14 side) are in LISTed to disk format: **hamurabi.lst, binomial.lst,** and **grammar.lst** To retrieve them do this:

NEW
ENTER "«name»"

The programs work unchanged in both 0.14 and 2.0 COMAL on the C64 and C128. If you transfer the files to a compatible disk format, they also will run virtually unchanged in CP/M COMAL and both Mytech and Unicomal IBM COMALs. To transfer the files, we recommend *Big Blue Reader CP/M* (see last issue, page 4).

Since Feb 1984 we've published over 1,300 pages for you in 18 issues of *COMAL Today*. We have maintained a very uneven publishing schedule, averaging 5 issues per year. No matter how hard we try to do 6 per year, delays crop up.

CP/M COMAL is now shipping ... so is its RUNTIME compiler. To celebrate shipping the first COMAL compiler we are including it in our subscribers only specials. See page 77.

National COMAL meetings are continuing on QLink every first Sunday and every second Thursday at 10pm Eastern time in our conference room. To get there go to CIN, then Magazine Rack, then COMAL Today, then Conference Room. **Meeting schedule is:**

QLink COMAL Meetings	
Sundays	Thursdays
August 2	August 13
September 6	September 10
October 4	October 8
November 1	November 12
December 6	December 10

We also maintain an active message base on QLink. If you have a COMAL question, check the posted messages, one may already help you. If not, just post your question. Come back a few days later, and you should have an answer (or two). While you are in the COMAL message base, if you have an answer for a posted question, please post a reply.

It's always nice to hear about how you use COMAL. We think COMAL compares well with other languages. We get many letters confirming this. We are reprinting two on page 9.

As soon as this issue goes to press, we have two major projects: new books and new demo disks. The release of several of our July 87 books has been delayed, hopefully only into August. The new Demo disks are desperately needed. The C64 COMAL 0.14 disk will include the new booting system, help files, and a good selection of programs and text! The CP/M demo will be updated with better programs. And the Mytech demo will hopefully have their updated IBM COMAL along with some better programs.

See page 77 for subscriber specials! ■

Special

You and Your Local School

Over 200 schools are using COMAL. If schools in your area aren't using COMAL yet, help them out. Tell them that COMAL is designed especially for schools ... and that COMAL is the official educational language in five European countries! With the new COMAL implementations coming out, schools can run COMAL on any computers it uses. While COMAL does make it easier for the teachers, it is the students who benefit the most. Let's give the students the best!

School COMAL Demo Pak

To help you convince your school to use COMAL we have a special **School COMAL Demo Pak**. It includes everything the school needs to preview COMAL:

C64 COMAL 0.14 Demo Disk
CP/M COMAL 2.10 Demo Disk
Mytech IBM PC COMAL 2.0 Demo Disk
Full set, **COMAL Today** issues 6-18
COMAL Today - The Index
5 copies of 24 page **COMAL INFO booklet**

All this for any school for **just $10** (prepaid only, shipping included to USA or Canada). We can send the package to you, and you can take it over to your school in person. Or you can give us the proper persons name and school address and we will send it to them for you. Or, the school can get the package themselves. In any case, they can try out COMAL on a C64, C128, CP/M or IBM computer. Plus, they will have **well over 1,000 pages** of COMAL information to read.

Once they start teaching COMAL, we will give the school a free subscription to *COMAL Today* (one per school, USA only) and quantity discounts on most items. We even let schools use a purchase order if they promise to pay the bill within 30 days! ■

Vote

VOTE #18

Please send us a postcard with your vote:

1) **Your favorite two articles in this issue?**

2) **What COMAL do you use?**

3) **Should we print rumors & advance info?**

VOTE #17 RESULTS

Your favorite article in *COMAL Today #17*:

#1 - Looking at Sprites - page 20
#2 - QLink Message Base - page 6
#3 - COMAL Kernal 1985 - page 40

What COMAL do you use?

COMAL 0.14	■■■■■■■■■■■
COMAL 2.0	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
SuperChip	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
IBM	■

Do agree with switching comma & semi-colon?

Yes	«none»
No	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Undecided	■■■■

What should "" IN "abc" return?

False	■■■■■■■■■■■■■■■■■■■■■■■■■■■■
True	«none»
3	«none»
4	■■■
-1	■■■
Undecided	■■■■■■■■■

What should KEY$ return for no key pressed?

Null	■■■■■■■■■■■■■■■■■■■■■■■■■
CHR$(0)	■■■■■■■■■■■■■■■■
CHR$(-1)	«none»
Undecided	■■■■■■

Guest Editorial

by Robert Ross

Reading the article in *COMAL Today #15* on the 1986 COMAL Standards Meeting, I realized that no one invited me. Boy, am I miffed. No matter that I probably would have declined an invitation with some all-wet excuse such as Denmark is a long way to swim and plane tickets cost money. But if I had been invited and if I had shown up and could listen dansk, I'm sure I would have enjoyed the debate of *"What value should be returned by the expression a$ IN b$ when a$ is the empty string?"* If I spoke dansk (I rarely USE dansk), I might have even joined in. After all, I was co-captain of the debate team my senior year in high school. No matter that the coach didn't actually let me debate that year. Encouraged by David Stidolph's question in *COMAL Today #16*, *"...what should "" IN "abc" return...,"* here goes.

Obviously, the expression should refuse all values that are incorrect and complain about poor quality control. Some of the confusion in this matter occurs because the operator in the expression is a real *smoothie*: it attempts to provide the answer to several questions at the same time. One has a TRUE/FALSE answer: is a$ a substring of b$? Another is: what is the character position in b$ of the first character of the first occurrence of a$ in b$, provided that a$ is a substring of b$?

A function to answer only the first question could be implemented several ways. The function could always return 0 for FALSE and 1 for TRUE. Or perhaps the function could return a count of the number of times a$ occurs in b$, such as 3 for "a" IN "abacad". But still the question of a$="" remains. The practical person might ask, *"Why do a test on something that is always true? You may as well program like:"*

```
IF a$="" AND NOT a$ IN b$ THEN
  REPORT 51,"system error in truth, forsooth."
ENDIF
```

On the other hand, the impractical person, especially one who understood set theory, might respond, *"The empty string is a substring of every string and must be reported as such: Tell it like it is!"* If this isolated substring function were to be a keyword, I would vote for an operator keyword called:

is'not'the'empty'string'and'is'a'substring'of

but with a much shorter name, such as..such as?

The empty string would also present a problem for a function to answer just the position'of'a'substring'in part of IN. **More people would affirm that "" is a substring of every string than would be willing to assign a character position to it.** If you allow LEN(b$)+1 why not LEN(b$)+2 also, not for the function value, which represents only one character position, presumably the first, but as the position of another appended ""? If two, why not an infinite number of empty strings appended to the end of every string? And a similar number preceding the string! What then of LEN(b$)? Should it always return the dimensioned length of b$? Or how about *overflow* every time it's used? No! Assigning a character position to "" is at least as bad, perhaps worse, than failing to report it as a substring.

The position'of'a'substring'in function also has a problem when the first string is not a substring of the second string, which is probably why it was originally combined with the TRUE/FALSE question for the operator IN: what character position can be returned if the position is requested when it doesn't exist? A *fictional position* error could be reported, but sometimes it is preferable to return a value that can be tested rather than to generate an error. Recognizing that IN is a combination of functions, I vote for the hodgepodge I have implemented as a function position'in.

more»

```
func position'in (a$,b$) closed
  if a$="" then
    return 0
  elif not a$ in b$
    return false
  else
    return a$ in b$
  endif
endfunc position'in
```

It returns zero as the position of the empty string in the second string; this is just as valid as LEN(b$)+1. It will return **FALSE** if the first string is not a substring of the second. It will return the character position in the second string of the first character of the first occurrence of the first string in the second when the first string is a substring of the second. By inspection of the function, it should be obvious that **IN** is a better name for this operation than **position'in**.

If this function doesn't replace the current definition of **IN**, my compromise vote is for a possibility that wasn't on the list in the article about the Standards meeting: "" IN "abc" could return -1. Negative values test true, to satisfy theoretical people, and in situations like:

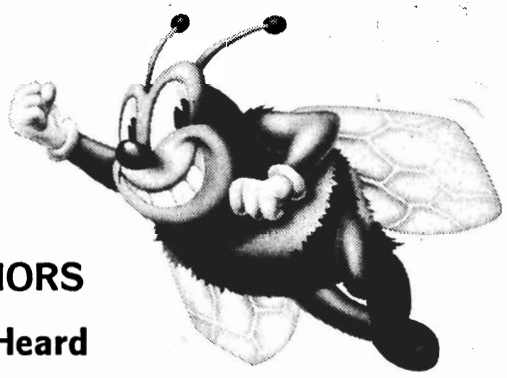
```
REPEAT
 a$=KEY$
UNTIL (a$ IN b$)>0
```

the last line is easier to type than:

```
until a$ in b$ and (a$ in b$) < len(b$)+1
```

Note that both versions obscure part of what is being tested and leave something to be desired in ease of readability and understanding.

Much thought has gone into COMAL standards. And the continuing discussions may be long, but they don't have to be acrimonious: COMAL lets you easily define a new function or re-define an old one to get exactly what you need. ■



RUMORS

We Heard

by Captain "*Buzz*" COMAL

Mytech called to tell us that they have corrected the bugs in their **IBM PC COMAL** and expect to release the **final in August 1987.** Expect the selling price to be **under $100.**

Mytech also said they have a **COMAL up and running on an Amiga.** They are testing it now for release in **September 1987.** They expect the price to be **under $200.**

Last issue we told you about the **CP/M COMAL RUNTIME "compiler"** system. It has been **shipping for over a month now.** We even have a special on it (see page 77). Now it seems that a **cross-compiler** is under way for the IBM and Amiga. **Apple COMAL** is designed with a compiler in mind as a **future option.**

The **MacIntosh COMAL** project may be revived soon. We have a test version running on a Mac here, and hope it will be finalized. But an **Atari ST** COMAL may be out before that!

COMAL was originally designed **specifically for use by schools.** By the end of the year it should be running on most of the computers used by schools: PET, Apple II, C64, C128, CP/M, IBM, Amiga, Atari, and MacIntosh.

Meanwhile, we hear that Mytech is working on a special version to **run on a college mainframe computer.** Soon students can learn COMAL on an Apple II or IBM PC **at school,** and run the same programs on their C64 or C128 **at home.** Later, **in college,** the programs will run on the schools mainframe. ■

How To ...

Submit Articles and Programs

If you have any COMAL information, programs, or articles that you would like to share, send them to:

COMAL Users Group, U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

If you submit a program, please send it on disk. A printed listing of the program is not necessary. If possible, also include a text file explaining the program. **Put your name as a remark at the beginning of your programs.** This helps us give proper credits if they are used. Most important: label the disk with your name, address and date. Also include disk format: C64, IBM, CP/M, Apple, etc.

Articles should be submitted as standard text files on disk. If possible, also include a printout of each text file on the disk. Don't include any special formatting commands in your files (we have to delete them). We use special formatting for our LaserJet printer. (Currently we use Word Perfect 4.2 on our Zenith IBM compatible system).

Don't worry if you aren't a professional writer. We try to keep this newsletter informal. The information and programs are more important than perfect grammar. Articles sent to us go through extensive editing. We actually go through over 4,000 sheets of paper while preparing one 80 page newsletter! You don't have to follow a bunch of rules, either. We rework your submissions to fit our newsletter format.

Material submitted is not returned. However, if you send us a disk, we will send one of our User Group disks back to you in exchange. Just specify which one. ■

Type In Programs

Line numbers are required for **your** benefit in editing a program (but are irrelevant to a **running** program). Thus line numbers usually are omitted when listing a COMAL program. It is up to **YOU** to provide the line numbers. Of course, **COMAL can do it for you.** Follow these steps to enter a COMAL program:

1) Enter command: **NEW**
2) Enter command: **AUTO**
3) Type in the program. When done:
   C64 COMAL 0.14: Hit «*return*» key twice
   C64 / C128 COMAL 2.0: Hit «*stop*» key
   CP/M COMAL 2.10: Hit «*esc*» key
   IBM PC COMAL: «*control*»+«*break*»
   Mytech: «*control*»+«C»

You may use both UPPER and lower case letters while entering a program. COMAL automatically makes keywords UPPER case and variable names lower case. (Note: C64 COMAL 0.14 may only use unshifted letters.) Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures, and COMAL will insert them for you. You **DO** have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add *//wrap* at the end. You must type it as one continuous line.

Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789'] [\ _

The «*left arrow*» key in the upper left corner of the C64/C128 keyboard is valid. COMAL 2.0 converts it into an underline. The C64/C128 computers use a «*British pound*» symbol in place of the \ backslash.

COMAL 0.14 users should read the notes on the next page.

more»

File Names



COMAL 0.14 Special Instructions

C64 COMAL 0.14 was designed to be an introduction to COMAL. As such, it does not implement the full COMAL Kernal and is lacking some common enhancements. However, many COMAL 2.0 programs will also work in COMAL 0.14 with only slight modification. Here are some tips:

PAGE is not implemented. Add this procedure:

```
proc page
  print chr$(147),
endproc page
```

CURSOR, INPUT AT and **PRINT AT** are not implemented. Add this procedure:

```
proc cursor(row,col)
  poke 211,col-1
  poke 209,(1024+(row-1)*40) mod 256
  poke 210,(1024+(row-1)*40) div 256
  poke 214,row-1
endproc cursor
```

Then for **INPUT AT** and **PRINT AT** first issue a **cursor** command, then use a normal **print**, **print using**, or **input** statement. Example:

```
  PRINT AT 9,1: USING "###":number; // 2.0
```

changes to:

```
  cursor(9,1)
  print using "###":num;
```

Additional procedures and functions that emulate COMAL 2.0 keywords are printed in past issues of *COMAL Today*. Over sixty procedures and functions are listed in *COMAL Today #15* starting on page 40. For example, if a COMAL 2.0 program includes **VAL** or **STR**, you can add a **val** function and **str** procedure as listed on page 49 in that issue.

Also, remember not to use SHIFTed letters when typing in a program or filename. ■

File Name Conventions

We try to include more than just COMAL programs on our disks. To help you determine a file type, we've adopted naming conventions.

All computer systems, up to 8 character NAME:

Name	Meaning (all systems)
NAME	COMAL program (IBM & CP/M automatically tack on their own extension)
NAME.LST	Listed program / proc / func
NAME.DSP	Displayed program
NAME.EXT	External procedure
NAME.DAT	Data file
NAME.TXT	Text file
NAME.DOC	Documentation file
NAME.BAS	Basic program
NAME.RAN	Random file

Commodore only ... up to 16 characters:

Suffixed	Prefixed	Meaning (C64/C128)
NAME.PROC	PROC.NAME	PROC listed to disk
NAME.FUNC	FUNC.NAME	FUNC listed to disk
NAME.DAT	DAT.NAME	Data file
NAME.TXT	TXT.NAME	Text file
NAME.DOC	DOC.NAME	Documentation file
NAME.EXT	EXT.NAME	External proc/func
	SHAP.NAME	Sprite shape file
	IMAG.NAME	Sprite shape file
	FONT.NAME	Font file
	SET.NAME	Basic type font file
NAME.BAT	BAT.NAME	Batch file
	SNG.NAME	Song file
	HRG.NAME	Color COMAL picture
NAME.HRG		Black/White bitmap
	CRG.NAME	Compact Color picture
NAME.CRG		Compact B/W picture
	ICON.NAME	Print Shop Icon file
	SCRN.NAME	Text Screen file
	POP.NAME	Mergeable Popover
NAME.POP		Program with Popover
	CHIP.NAME	Super Chip Program ■

Questions & Answers

A GOOD CASE

Does COMAL have an equivalent to BASIC's ON...GOSUB? - B K Ball, Torrance CA

The COMAL CASE structure is an improved ON...GOSUB. It keeps the sections together in the CASE structure, rather than GOSUBing all over the program. CASE has many advantages over ON...GOSUB. For example, if **x** *is your control variable, BASIC can only deal with it sequentially. It can't handle possible values of 1, 2, and 8 easily. CASE can. CASE also can use a string as its control variable. Finally, several values that need the same result can be grouped into one CASE, but would require a separate entry with ON...GOSUB. Here is a part of a program demonstrating CASE:*

```
LOOP
  PRINT "1. Add"
  PRINT "2. Subtract"
  PRINT "3. Quit"
  INPUT "What task?": task$
  CASE task$ OF
  WHEN "1","A","a","Add"
    add
  WHEN "2","S","s","Subtract"
    subtract
  WHEN "3","Q","q","Quit"
    EXIT
  OTHERWISE
    PRINT "What?"
  ENDCASE
ENDLOOP
```

This example shows how you can print a menu on the screen, and allow several possible replies trigger a task. The user could type in the number 2 or the letter S and get the subtract task. Also, notice the OTHERWISE section provides a way to deal with a condition that does not match any of the cases.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

The : Colon

Sid Seiferlein of Muskegon, MI wants to know the meaning of a colon between XU and +YU in the statement: **XU:+YU**

The line above is an incremental addition. You were confused because you took the plus sign as part of the variable **YU.** *Actually, the colon is not a separate entity in the statement. The* **:+** *is one item meaning increment by. For example, the following two statements are equivalent:*

```
TOTAL:=TOTAL+AMOUNT
TOTAL:+AMOUNT
```

Both of these lines will add **amount** *to* **total.** *Thus, if* **total** *starts at 15, and* **amount** *is 5, after the statement is executed,* **total** *equals 20. Decrementing a variable is similar. The following two lines are equivalent:*

```
MONEY:=MONEY-BET
MONEY:-BET
```

This will subtract the value of **bet** *from* **money.** *Thus, the* **:+** *and* **:-** *operators are shortcuts.*

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

PLOTTEXT

After **SETGRAPHIC 1** the **PLOTTEXT** command is not very readable. Is there a way to have legible text on the multi-color screen? I use my special characters for this screen now, but it wastes memory. - Sid Siferlein, Muskegon, MI

The **SETGRAPHIC 1** *multi-color graphics screen uses pairs of pixels to display a single color. This is fine for drawing. But* **PLOTTEXT** *is designed for hi-res operation. The solution is to use a special charater font, as you discovered, or to avoid using* **PLOTTEXT** *on the multi-color screen.* ■

Letters

QLink E-Mail from: GlynnS -- some comments about CP/M COMAL and the new documentation packages. I downloaded the CP/M COMAL demo *[from QLink]* and like it. I like the new TRACE option. I'll be waiting for the regular version when it comes out. *«1»* I have not figured out how to exit COMAL to CP/M yet. *«2»* I'm a programming language nut and play with CP/M mode because more languages are available there than in the native Commodore modes.

I was interested in using 16 bit port addresses because that is the way you have to access the I/O chips to do sound and graphics in C128 CP/M. Turbo Modula-2 is the only one I've found (high-level language that is) that allows me to do that in CP/M. *«3»*

The strongest point in COMAL's favor is immediate syntax checking upon line entry. Other languages attempt to make program development easier with memory resident editors and compilers (Turbo Pascal) or use RAM disk (Abacus Software) or have syntax checking in the editor (KMMM Pascal). None have syntax checking on line entry. I've seen many people become frustrated with BASIC for that reason.

Some languages have features that I still like to use that are not available in COMAL such as records and pointers, extensive exception handling *«4»* and separate compile *«5»* and linking capability. But this takes you beyond a language for learning. I still believe COMAL is the best language for learning and playing around with ideas. One can still use other languages if they want to use options not available in COMAL. I don't believe there is one best language for everything. You have to use the language that best helps you with the problem at hand.

Thanks for the comments.

«1» The CP/M COMAL 2.10 is final and is now shipping, as is the full RUNTIME "compiler" system.

«2» To exit COMAL back to CP/M use the command: **BYE**

«3» CP/M COMAL also can play with the I/O chips with the INP and OUT commands. We also have procedures which can PEEK and POKE RAM bank 0 and the VDC chip from CP/M. Super Chip for the C64 cartridge can access the C128 VDC chips.

«4» COMAL 2.0 has an advanced error handling system... the TRAP / HANDLER structure.

«5» Both CP/M and IBM COMALs now have a runtime compiler system.

¤¤

Dear Editor: COMAL is a very useful language. You tend to push it as an alternative to BASIC and an ideal educational language for schools. It is better than that. I think it is an excellent development language. It's very friendly, like FORTRAN with a good debug mode. BASIC is a cruel hoax as a programming language. It made sense only when home computers had very small memories. There is no need for BASIC in computers with 64k or more memory. I recently acquired Oxford Pascal and found nothing to recommend it over COMAL except that universities are stuck on it.

I bought my C64 3 years ago as a disposable computer. I thought that within two years the 16 bit machines would stabilize and come down in price and I would discard the C64. I did little initial programming since BASIC 2.0 was so useless. I have been using COMAL with the C64 for two years now even though 16 bit computers are affordable. While I occasionally lust for more memory, fast disk access, and a few commercial software packages, I seem to be getting by with COMAL on the C64. I would not consider buying a computer that did not have the COMAL 80 language.

Incidentally, one of the best features of COMAL is *COMAL Today* and the quality and diversity of the people who support it. - Alan Jones ■

COMAL Clinic

SIMPLE LOGIC

Dick Klingens reminds us that in COMAL it is possible to use relational operators in assignment statements:

```
a:=19
b:=87
c:=(a>b)
```

The logical expression a>b is evaluated into its logical value:

A is not greater than b, so the value of a>b is FALSE. c is set to FALSE (zero).

Knowing this, we can define:

```
FUNC max(x,y) CLOSED
  RETURN (x>=y) * x + (x<y) * y
ENDFUNC max
```

Using the same values (19 and 87) for a and b:

```
PRINT max(a,b) // gives
87
```

Explanation: The values for x and y in the function are

```
x:=19
y:=87
```

Now, the logical value of x>=y is FALSE (zero) and the value of x<y equals TRUE (one). So the value returned by the function is:

```
FALSE * 19   +   TRUE * 87
```

or:

```
0 * 19   +   1 * 87
```

and that equals 87.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

TRANSLATE VIZA WRITE FILES

Read'viza-file on *Holland Disk #7* is a simple utility from Dick Klingens to translate *VizaWrite* files to normal Commodore ASCII. It can translate to a file, screen, or printer.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

FIND & DEFKEY COMBO

Sometimes you have many similar lines in a program listing. You might wish to change them into a **procedure** call. For example:

```
INPUT AT 24,1,0: "Hit <return> to continue": z$
```

Here is the **procedure** to add to the program:

```
PROC continue
  INPUT AT 24,1,0:"Hit <return> to continue":z$
ENDPROC continue
```

And this is an easy way to change all those similar lines into a procedure call:

```
USE system
defkey(7,"continue"11""13"")
FIND "INPUT AT 24"
```

Now, each time the line is found, just press «*F7*» to change it. Press «*return*» to not change it. We are using a neat trick in the second line. The "11" is a CHR$(11) which will erase to the end of the line. The "13" is a carriage return.

This method is general enough that it will let you change lines similar but not identical to the original one. For example, it also finds:

```
INPUT AT 24,1,0: "Press RETURN to go...": ret$
```

The CHANGE command can't do that!

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

more»

GEMINI 10X

Bob Hoerter informs us that «*control*»+D screen dump will work on the Gemini 10X printer with the Xetec Super Graphics interface if the following set up is made on the interface:

SET in TRANSPARENT mode
SET printer as Epson 132

This will give a smooth printout of the graphics screen on the right side of the page.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

FLEXIBLE ENDING CP/M COMAL

CP/M COMAL provides several ways to have your program end. If you don't include an **END** statement, when the program ends, COMAL issues a carriage return and then prints:

End of program

If you include a simple **END** statement, COMAL leaves the cursor where it is when the program ends and does nothing. However, if you include a null string after the END like this: **END ""** COMAL will issue a carriage return after the program ends. Finally, you may include a message after the END like this: **END "done"** Now, COMAL prints the message starting at the position of the cursor when the program ends.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

Super Chip C128 Warning

Warning: Super Chip uses the cassette buffer for some of its C128 features. This may conflict with programs that need this buffer for other purposes. You can stop Super Chip from using this buffer. Just type:

```
USE c128
discard'c128
```

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

COMAL COPYSCREEN

On page 34 of *COMAL Today #13* Patrick Roye gives a procedure to print a user font style text screen to a printer. David Stidolph converted it to copy the text screen to the graphics screen. Now, Dick Klingens gives us a procedure without PEEK or POKE to do it:

```
PROC copyscreen CLOSED
  DIM line$ OF 40
  USE graphics
  graphicscreen(0); clearscreen
  OPEN FILE 99,"ds:",READ
  FOR row#:=1 TO 25 DO
    CURSOR row#,1
    INPUT FILE 99: line$
    plottext(1,200-row#*8,line$)
  ENDFOR row#
  CLOSE FILE 99
ENDPROC copyscreen
```

The above procedure is very fast. It accurately copies the normal and shifted fonts. It does not accuratly copy reversed characters and has the side effect of scrolling the textscreen.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

COMAL CORNER

Joel Rea wrote a nice interactive COMAL Corner for LOADSTAR disk. It on the back side of *LOADSTAR* disks #9, 12, and 13 under *Paperless Pages.*

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

SEIKOSHA GP-550A PRINTER

Morgan Allsup, PO Box 339, Pinon, AZ 86510 is looking for a graphic screen dump routine for this printer. Please contact him if you can help him get a screen dump. ■

Bug Fixes

LOAD / SAVE Bug Fix

Page 50 of *COMAL Today #7* reports a problem with the **LOAD** routine when packages and either sprite shapes or a font are linked to a saved program. When the program is loaded, the file is closed after the package data has been loaded. But this is before the font or shape data can be loaded. All the linked material is in the file, but the keyboard is used to try to input the font or shapes!

After additional study, Robert Ross has shown that **LOAD** will load programs with mixtures of linked material if the package libraries are written after all of the other linked material. Unfortunately, **SAVE** writes all of the linked libraries first.

The program *re-linker* reads a saved program file and writes a new program file of the same length which **LOAD** can handle. The program also generates a listing of the locations where the linked material was saved from.

To use *re-linker*, first **SAVE** a COMAL program with linked material (example *flexi*). Then **LOAD** and **RUN** *re-linker*. The program first asks for the name of the saved program: (*flexi* in our example).

Next it needs a file name to use for the fixed program file. If you don't want to create a fixed file, just press «*return*». Using our example file, you might reply: *new'flex*.

Finally, information about the program file being fixed is generated. A final prompt asks for a file name for this data. Reply **ds:** for screen output, or **lp:** for printer output. If you don't want to see this data, just press «*return*».

[*We used re-linker to allow Bill Nissley's comal-flex programs to include fonts and a package used to rotate them.*]

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

0.14 BOOT BUG

In *COMAL Today #9* we introduced Phil Bacon's *ml.sizzle* fast loader program for COMAL 0.14. Most COMAL users were very pleased to have the load time reduced from over 90 seconds to less than 30 seconds. However, users with 1541 compatible drives (like the *Enhancer*) reported problems with *ml.sizzle*. We released a new updated *ml.sizzle* file on *Today Disk #17*. Then several owners of 1571 disk drives reported a similar problem. This came as a complete shock to us because it works fine on our 1571 disk drive. However, we verified this problem at the MARCA Commodore Show when our Demo Disk would not boot on the 1571 drive Commodore provided us.

Our solution is to include two boot files on the disk and to revert back to the original *ml.sizzle*. Our standard *boot** file has been changed so it does not use a fast loader. We have added a *fast** file to our disks so that 1541 and 1571 users can still make use of the fast loader. Now, the first time you load COMAL with the usual: **load "boot*",8.** It will load COMAL in the normal manner which will work with all 1541 compatible drives. It also prints a message on the screen telling how to use the fast boot to load COMAL next time. To do so you use: **load "fast*",8.** (After loading either file, you must type **run**.)

If for some reason you cannot get the boot program to work on your COMAL disk, you can load COMAL directly with the command: **load "c64 comal 0.14",8**

This will not automatically **CHAIN** our *hi* program, expand program memory, or place error messages in RAM, but it will allow you to use COMAL until you can get a current disk.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

more»

EDIT RANDOM FILE

Edit'wheel'data on page 55 of *COMAL Today #15* has two incorrect **TAB** statements. The fourth line of the program should read:

```
print tab(19);"ran.wheel"+chr$(142)+chr$(8)
```

The fifth line in **write'data** should be:

```
print tab(10);filename$
```

The program on *Today Disk #15* is correct.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

SPRITE BUG

Robert Ross found that if the number of jiffies given in a **MOVESPRITE** command is an integer multiple of 256 (for example: 256, 512 or 768), the sprite won't start to move. Fortunately, a sprite that is moving does not stop when the number of jiffies remaining gets to a multiple of 256, so it is not hard to get around this problem. Just use a value one higher or lower when giving the **MOVESPRITE** command.

¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

EXTENDED PRINT USING

The **print'using** procedure from *COMAL Today #16*, page 39, allows you to insert commas into large numbers being displayed. For example:

```
print'using ("#,###",1000)
1,000
```

The comma above is required:

```
print'using("####",1000)
```

produces an error. This bug has been carried over to our CP/M COMAL demo disk. Any help solving this problem would be appreciated. ■

Cryptograms



By Bill Inhelder

Puzzle pages in newspapers and magazines often have cryptograms. A cryptogram is a statement, usually by a famous person, which is coded by means of a simple replacement code. The object of the puzzle is to decipher the message. Some cryptograms are relatively easy while others are devilishly difficult, requiring careful study, intuition, perseverance, luck and a good eraser. The recording task can be messy and time consuming. Suppose you decide that the D is an A and you then write an A under each D in the message. Later it turns out that the D should be an O. All the A's must be erased and replaced with O's without erasing nearby letters. A computer can expedite this task.

The program *cryptogram* on *Today Disk #18* is designed both as an aid in housekeeping as well as analysis. It provides a table of the frequency distribution of the letters in the message. This can be helpful when coupled with the knowledge that the most frequently used letters in English are: e, t, a, o, n, r, i, s. Another helpful aid is a list of unused letters. All this in no way replaces the human skill in analyzing the punctuation and the short words for hints as to how to proceed.

The cryptogram may be entered from the keyboard or the program *encrypt* can be used to write a cryptogram to a disk file for later use. The message may only be blanks, capital letters, and the punctuation , ; : ! ? " - ' . An example data file named *crypt2.dat* is also included on *Today Disk #18* to get you started.

More Cryptograms

Bob McCauley also sent us a cryptogram program. *Crypto'solver* is on *Today Disk #18*. Now you will be able to compare two different ways of handling cryptograms with your computer. Perhaps you can write your own cryptogram program after looking at them. ■

CP/M COMAL



CP/M COMAL 2.10 will run on any CP/M machine under CP/M 2.2 or CP/M 3.0. It is final and being shipped. There are three main items: a demo version, the full version, and a RUNTIME "compiler" system. See page 77 for our special introductory offer on CP/M COMAL.

The CP/M COMAL demo version may be freely copied and distributed. It is the full COMAL implementation, except the **SAVE**, **MERGE**, and **ENTER** commands are disabled. With it you can **LOAD** and **RUN** any CP/M COMAL program. You can type in your own programs and **LIST** them to disk or printer. This allows you to try programming on a full COMAL 2.0 system without making a large investment.

The full COMAL system comes with a 170 page manual in a **Doc Box**. (*This full version may not be given away.*) It runs well on the C128 in CP/M mode. (*It works faster on other CP/M computers such as a Kaypro.*) It supports the 80 column text screen and uses the RAM expander, if available, as a RAM disk. If you own a C128 and want to move up from COMAL 0.14 to 2.0, CP/M COMAL may be a good choice. CP/M COMAL is a full implementation of COMAL 2.0, but lacks some common extensions such as graphics and sound (currently in the works).

CP/M COMAL is our first COMAL to offer a RUNTIME "compiler" system. RUNTIME attaches a support system to any SAVEd COMAL program file. The result is a stand alone CP/M program file which no longer needs the COMAL system to **RUN**. (As with the full COMAL system, the RUNTIME system may not be given away or sold. However, the stand alone files it creates can be distributed royalty free.)

The file generated by RUNTIME has a .COM extension (such as **tower.com**). The .com program is run simply by putting the disk in the drive and typing the name of the program from CP/M mode. A CP/M COMAL program runs at the same speed under RUNTIME as normal.

CP/M COMAL Support

We are in the process transferring existing COMAL programs onto CP/M disks. Hopefully we soon will have CP/M COMAL program disks available. Also, many of the programs in *COMAL Today* will run under CP/M COMAL. We are not yet ready to have a CP/M format *Today Disk*, but hope to soon. Any CP/M COMAL programs from our readers will be appreciated.

Another area of support is for machine language packages under CP/M. We can't offer advice about Z80 code, but we can help in other ways. In particular, Ray Carter has already sent us a preliminary graphics package for the C128. When finished, this will offer commands to draw with 640 by 200 resolution. A side effect of this project is likely to produce a font package.

In COMAL, if a program line is too long to be listed on a single screen line, the overflow from the first line is continued on the next line, and is indented the same amount as the first line. This aids readability. CP/M COMAL also does this for a program line listed to disk. This is great if you will use the file with a word processor. But a problem comes when the line is entered back into COMAL. If there are spaces in the middle of a keyword or variable name, the line is not accepted by the parser. This does not apply to programs SAVEd to disk, or programs that do not contain long lines.

There is a quick way to get around this. We just trick COMAL into thinking it has a larger screen. This is done by a **POKE** to location 539. After the program has been listed to disk, you should restore this location back to the default 80 columns. For example, to **LIST** the program, *myprog* to disk, issue these commands:

```
POKE 539,200
LIST "myprog"
POKE 539,80
PAGE
```

more»

CP/M Keys



CP/M TRACE Command

CP/M COMAL has provided a **TRACE** command that is different from that of the COMAL cartridge. In CP/M, **TRACE** prints each line as it executes, as well as results of calculations and the initial values of parameters for procedures and functions. **TRACE** may direct output to the screen **ds:** or the printer **lp:**. **TRACE OFF** is used to exit **TRACE** mode, so it is possible **TRACE** selected portions of a program without tracing the whole thing. The following program demonstrates the **TRACE** command.

```
0010 TRACE "lp:"
0020 PRINT gcf(9,12)
0030 TRACE OFF
0040 //
0050 FUNC gcf(x,y)
0060   IF x MOD y=0 THEN RETURN y
0070   RETURN gcf(y,x MOD y)
0080 ENDFUNC gcf
```

Here is the output as listed to the printer. Our comments are in small print.

```
0020 PRINT gcf(9,12)
<9><12> «-- 9 and 12 are parameter values
0060 IF x MOD y=0 THEN RETURN y
<0> «-- ((9 MOD 12)=0) is FALSE <0>
0070 RETURN gcf(y,x MOD y)
<12><9> «-- 1st recursive function call
0060 IF x MOD y=0 THEN RETURN y
<0> «-- x=12, y=9, (12 MOD 9)=3 not 0
0070 RETURN gcf(y,x MOD y)
<9><3> «-- 2nd recursive function call
0060 IF x MOD y=0 THEN RETURN y
<1><3> «-- expression is true <1>, y=<3>
0080 ENDFUNC gcf
<3> «-- 2nd recursive call returns <3>
0080 ENDFUNC gcf
<3> «-- 1st recursive call returns <3>
0080 ENDFUNC gcf
<3> «-- main function call returns <3>
0030 TRACE OFF ■
```

CP/M as supplied for the C128 does not support all the keys on the keyboard. In particular, the lower cursor keys don't work and «*insert*» does a «*delete*». COMAL has overcome this problem.

Keysc128.sav on the *CP/M COMAL DEMO disk* defines those keys as expected by Commodore users. Once the program is run, use *KEYFIG* to make those keys active each time you boot CP/M. Follow these steps:

1) Boot CP/M. Insert the COMAL demo disk and type next to the prompt, A>
   A> **demcomal**
2) from COMAL, type:
   run"keysc128"
3) Now execute the **c128** procedure. Type:
   c128
4) You get an information screen. Hit «*return*» one more time.
5) Exit COMAL. Type:
   bye
6) Insert a disk with *KEYFIG* on it. (*KEYFIG* comes on the C128 CP/M system disk. It is not on the *CP/M COMAL Demo Disk*.) Type:
   keyfig
7) Respond to the help prompt with
   n
8) Cursor down twice to the current definitions prompt and hit «*return*».
9) Cursor down twice more to the exit prompt and hit «*return*» again.
10) Insert your CP/M boot disk and hit «*return*» by the prompt to save the file on the CP/M boot disk.
11) To exit the program, type
    n

Your boot disk is now corrected and will allow you to use the extra keys with CP/M COMAL. You never need to repeat this process again. Experts may want to use *KEYFIG* to define the function keys for common commands such as **CAT, LOAD, SAVE,** or anything else. *KEYFIG* also lets you define the screen colors, but that is up to you. ■

Mouse Package 

by Garrett Hughes, David Stidolph, Richard Bain

The Commodore 1351 mouse plugs into the joystick port. The **mouse** package detects its movement as you roll it across a flat surface.

The package starts at location $7f00 (reducing program memory by 256 bytes). It must start before the cartridge because it uses the 60 times per second interrupt of the C64 to continuously update the mouse position. It works with Super Chip, but not with packages located at $8009 nor with the **font** package.

This package moves an arrow pointer on the graphics screen matching the motion of the mouse. The buttons on the mouse are detected and the pointer can be turned on or off. To use the mouse, issue these commands:

```
LINK "pkg.mouse"
USE mouse
```

The pointer is automatically displayed on the graphic screen and will move as you move the mouse! But, you must switch to the graphics screen before you can see it. However, all **mouse** commands are active from the text screen as well as from the graphics screen (you just can't see the pointer there).

Showmouse - makes the mouse **arrow** visible
Hidemouse - makes the mouse **arrow** invisible
Mousex - returns the current x co-ordinate of the mouse (from -24 to 321)
Mousey - returns the current y co-ordinate of the mouse (from -6 to 249)
Mousepos(«xcor»,«ycor») - moves the mouse to the specified x,y location
Buttonleft - returns **TRUE** if the left button of the mouse is currently pressed
Buttonright - returns **TRUE** if the right button of the mouse is currently pressed
Version'mouse$ - returns the version number and authors of the mouse package

By default, the mouse pointer uses sprite number 6 and image number 30 to provide a white unexpanded arrow that has priority over data. (It is black if the screen is white when the **USE mouse** is issued.) If this arrangement is not acceptable for your application, the following mouse commands may be useful, as well as sprite commands after you **USE sprites**:

Mouse'sprite - returns the sprite number being used for the mouse pointer (default is 6)
Identify'mouse(«sprite#») - sets the sprite number used as the mouse pointer (0-7)
Mouse'sleep - disconnects the interrupt driven routine controlling the mouse. To restart the mouse, issue the command **USE mouse** or **showmouse**.

The following program prints the current location of the mouse on the text screen:

```
USE mouse
LOOP
  CURSOR 10,10
  PRINT USING "-### -###":mousex,mousey
ENDLOOP
```

The next program allows the mouse to be used as a *pen* to draw with very smooth lines on the graphics screen. Press the left button to draw a straight line from the last point to the current position. Hold the left button to draw as you move the mouse, or press the right button to move without drawing. The **mouse package** and *demo/mouse* are on *Today Disk #18.*

```
USE mouse
USE graphics
graphicscreen(0)
LOOP
  IF buttonleft THEN
    drawto(mousex,mousey)
  ELIF buttonright THEN
    moveto(mousex,mousey)
  ENDIF
ENDLOOP ■
```

A Matter Of Style



[Programming can be seen as an art since it can be very creative, or as a science, since it can also be very precise. In practice, it is a little of both. Thus, we think programming can be a matter of style. We believe that COMAL Today should encourage good programming practices. Thus, even though COMAL includes a GOTO statement, we avoid using it in any published programs. We don't necessarily reject submitted programs using GOTO; we simply rewrite them using other COMAL structures. We feel that the end result is a more readable program that is easier to follow. The following article and program were submitted by Bill Inhelder. You have enjoyed his material before. We thought you would be interested in how his program evolved, along with his comments.]

Expanded Decimal Representation

by Bill Inhelder

Part of the enjoyment of writing a program to perform a specific task is in devising a method for carrying out that task. Sometimes the method is direct and obvious; however, most often it requires considerable thought and, on occasion, a bit of creativity. Not all methods or algorithms are based on some precise mathematical formula. Once the method of carrying out the task has been determined, the easy part is coding the method in some computer language.

Suppose the task is to represent all positive fractions in their decimal form. At first glance this might seem more appropriate for a calculator rather than a computer. We know that some fractions have finite decimal expansions while others have infinite repeating decimal representations. For example 17/64 = 0.265625 while 13/99 = 0.(13). The parentheses indicate that the pair of digits, 1 and 3, repeat indefinitely. We speak of the fraction as having a period of 2. We might also recall from our math classes that the largest period a fraction can have is one less than the denominator. Thus 1/7 has a period of 6 and is 0.(142857). However 3/43 has only a period of 21.

Why is a program necessary to carry out a simple division on either a calculator or computer? When I find the decimal equivalent of 1/13 on my calculator, it displays .076923077 because my calculator has 9 digit precision. I suspect that the period is .076923 and that it repeats following the last digit, but I cannot be sure. Had I entered 3/43, my calculator would be no help.

We require an algorithm which does not use division of real numbers and which will give as many digits as required by the fraction. The following method can be used.

If the numerator is smaller than the denominator, multiply it by 10 and perform integer division and remaindering using the **DIV** and **MOD** functions.

For example: 1/8 becomes 10/8 = **1 R 2**. Multiply the remainder by 10 and divide it by 8, 20/8 = **2 R 4**. Multiply 4 by 10 and divide by 8, 40/8 = **5 R 0**. The zero remainder indicates a finite decimal representation. Thus 1/8 = .125.

Since this algorithm can be repeated as many times as we wish (if **R** is not equal to zero), we are no longer limited by the precision of the computer.

Several problems still remain. How can we determine when the sequence of digits repeat, especially when there are several non-repeating digits before a sequence of digits begins to repeat? Example: 45/56 = 0.803(571428)

The solution to the problem lies in checking the remainder at each stage with the set of remainders which preceded it. If a match is found, then the digits must repeat from that point on. For example consider 279/1100.

more»

2790/1100 = 2 R 590
5900/1100 = 5 R 400
4000/1100 = 3 R 700
7000/1100 = 6 R 400

Thus at the 3rd stage we compare 700 with the two previous remainders and find no match. However, at the 4th stage when we compare 400 with the previous remainders we get a match at the 2nd stage. This implies that 279/1100 = 0.25(36).

Since the digit could repeat immediately (1/3=.(3)), the algorithm in the program compares the very first calculated remainder with the numerator which it treats as the original remainder. One further concern must be addressed. The program must provide for the calculation of an improper fraction which leads to digits to the left of the decimal point.

The implementation of the above algorithm in the program *decimal'rep* is somewhat complex because of the nested FOR loops and IF statements. Remarks have been added to help the reader follow the program instructions.

For non-programmers who are not interested in how an algorithm is used to carry out a task, the program *decimal'rep* can be used to predict when a fraction will have a finite decimal expansion and, if so, how many digits in length. Also when will a fraction have an infinite decimal expansion. If so, how many non-repeating digits are there and what is the length of the period of repeating digits.

The rule for determining the length of the infinite repeating sequence of digits is as follows: after the factors of 2 and 5 have been removed from the denominator, the remaining number must divide (exactly) the first number of the form 10^N-1. N will then represent the period of the denominator.

For example 140 = 2^2 x 5 x 7; try 3/140. 7 must divide either 9, 99, 999, 9999, etc. exactly. Since 7 does not divide any of the above numbers until 999999, the denominator 140 will have a period of 6 since 999999 = 10^6-1.

The rule is of theoretical interest only since it is impractical in most cases because of the size of the period. Running the program is the easiest way to determine the period.

Computer programs can offer different challenges to suit the varied interests of individuals. The desire for a more efficient algorithm to perform the task is the challenge of the programmer. To apply the program to discover something new is the challenge of the program user. Some people are attracted to the programming challenge; others to the application challenge; and still others to both challenges. *Decimal'rep* is on *Today Disk #18.*

Note to programmers: COMAL allows the use of the GOTO label statement. This is the first program where I have found it necessary to use the GOTO statement. If anyone can suggest a way to accomplish all the objectives of the program and not use the GOTO statement, I would appreciate hearing from you.

[The following listing is the program as sent to us by Bill Inhelder. A few places we have inserted a number inside « » brackets. At the end of the listing we have a note that applies to that section of the program.]

```
PAGE // decimal'rep'orig
PRINT "This program will output the decimal"
PRINT "expansion of all fractions which"
PRINT "repeat or terminate in less than 100"
PRINT "digits.  Because of the precision of  "
PRINT "the computer, the numerator and
denom-" // wrap line
PRINT "inator should be less than 10 digits."
DIM quotients(100), remainders(100)
PRINT "Repeating portion of decimal expansion"
```

more»

```
PRINT "is enclosed in parentheses."
PRINT
PRINT "Enter numerator, denominator:"
PRINT "Enter 0,0 to quit."
INPUT "": numerator,denominator
PRINT
WHILE numerator<>0 DO // «1»
  orig'numerator:=numerator;
  remainders(1):=numerator // wrap line
  IF numerator<denominator THEN numerator:=
  numerator*10 // wrap line «2»
  FOR i:=1 TO 99 DO
    quotients(i):=numerator DIV denominator
    // wrap line «3»
    remainders(i+1):=numerator MOD
    denominator // wrap line
    //terminating decimal expansion
    IF remainders(i+1)=0 THEN
      //proper fraction
      IF orig'numerator<denominator then//«p1»
        PRINT orig'numerator;" / ";denominator;
        " = 0."; // wrap line
        FOR k:=1 TO i DO PRINT quotients(k);
        PRINT
        PRINT
        GOTO again // «4»
      ELSE // «p2»
        //improper fraction
        PRINT orig'numerator;" / ";denominator;
        " = ";quotients(1);"."; // wrap line
        FOR k:=2 TO i DO PRINT quotients(k);
        PRINT
        PRINT
        GOTO again // «4»
      ENDIF
    ELSE
      //non-terminating decimal expansion
      FOR j:=1 TO i DO
        //matching remainders indicate the
        beginning of repeating digit cycle//wrap
        IF remainders(i+1)=remainders(j) THEN
          //for proper fraction
          IF orig'numerator<denominator THEN
            PRINT orig'numerator;" / ";
            denominator;" = 0."; // wrap line
            IF j=1 THEN // «p3»
              //no digits in non-repeating
              //portion of decimal
              representation // wrap line
              PRINT " (";
              //digits in repeating cycle
              FOR k:=j TO i DO PRINT
              quotients(k); // wrap line
              PRINT ")"
              PRINT
              GOTO again // «4»
            ELSE // «p4»
              //digits in non-repeating portion
              of decimal representation//wrap
              FOR k:=1 TO j-1 DO PRINT
              quotients(k); // wrap line
              PRINT " (";
              //digits in repeating cycle
              FOR k:=j TO i DO PRINT
              quotients(k); // wrap line
              PRINT ")"
              PRINT
              GOTO again // «4»
            ENDIF
          ELSE // «p5»
            //improper fraction
            PRINT orig'numerator;" / ";denomin
            ator;" = ";quotients(1);".";//wrap
            //digits in non-repeating portion of
            decimal expansion // wrap line
            FOR k:=2 TO j-1 DO PRINT
            quotients(k); // wrap line «5»
            PRINT " (";
            //digits in repeating cycle
            FOR k:=j TO i DO PRINT
            quotients(k); // wrap line
            PRINT ")"
            PRINT
            GOTO again // «4»
          ENDIF
        ENDIF
      ENDFOR j
      numerator:=10*remainders(i+1)
  ENDIF
ENDFOR i
PRINT "Fraction does not repeat in the first"
PRINT "100 digits." // «6» «p6»
```

more»

```
  PRINT
again:
  PRINT "Enter numerator comma denominator:"
  PRINT "Enter 0,0 to quit."
  INPUT "": numerator,denominator
  PRINT
ENDWHILE
END
```

These are the explanations for our notes in the program listed above:

«1» *no check is made for denominator=0, this could lead to a division by zero error later.*

«2» *this IF is unnecessary and leads to an ambiguity which later requires 6 print sections (see our «p» notes below). The first 5 print sections must check if the multiplication by 10 was made.*

«3» *the index into the two arrays named remainders and quotients are not synchronized. Quotients uses i while remainders uses i+1.*

«4» *these GOTO statements are not needed. See the revised program listing.*

«5» *it is possible for the FOR loop to be skipped entirely if i=2. This happens when there are no non-repeating digits before the repeating digit sequence begins.*

«6» *after calculating the fraction to 100 digits, it is not printed.*

These are our explanations about the 6 print sections in the program:

«p1» exact decimal fraction less than 1
«p2» exact decimal fraction greater than 1
«p3» repeating fraction less than 1 with no non-repeating digits
«p4» repeating fraction less than 1 with at least one non-repeating digit
«p5» repeating fraction greater than 1
«p6» fraction does not repeat within 100 digits

We sent the following note back to him:

This letter is inspired by your *decimal'rep* program. I like the idea, although I haven't had a chance to compare it with the routines used in the *calculate'pi* program. The trouble is, we have a policy not to release any programs with a GOTO statement.

I rewrote your program without any GOTO's. It is enclosed. I think you will find it a logical version. The only trick I used was in the first FOR loop in the program. I don't exit the loop when I find the decimals repeat. Instead, I take advantage of the fact that if I find one match, it will be impossible to find another. Therefore, I execute the loop a few times unnecessarily rather than continually testing if I can leave early. - Richard Bain, COMAL Users Group USA

```
PRINT "This program will output the decimal"
PRINT "expansion of all fractions which"
PRINT "repeat or terminate in less than 100"
PRINT "digits. Because of the precision of"
PRINT "the computer, the numerator and"
PRINT "denominator should be less than 10"
PRINT "digits."
PRINT "Repeating portion of decimal expansion"
PRINT "is enclosed in parentheses."
max'digits:=101
DIM q(max'digits), r(max'digits) // «1»
PRINT "enter numerator, denominator"
INPUT "enter 0,0 to quit: ": n,d // «2»
n:=INT(ABS(n)); d:=INT(ABS(d))
WHILE n<>0 AND d<>0 DO // «3»
  orig'n:=n; digit:=0; start:=0
  WHILE n<>0 AND digit<max'digits DO
    digit:+1; q(digit):=n DIV d // «4»
    r(digit):=n MOD d; n:=r(digit)*10 // «5»
    FOR x:=1 TO digit-1 DO
      IF r(x)=r(digit) THEN start:=x; n:=0 // «6»
    ENDFOR x
  ENDWHILE
  PRINT
  IF start THEN // «7»
    PRINT "number repeats"
  ELIF n=0 THEN
    PRINT "number is exact"
```

more»

```
  ELSE
    PRINT "number does not repeat in";
    PRINT max'digits-1;"digits."
  ENDIF
  PRINT orig'n;"/";d;"=";q(1),".", // «8»
  IF start THEN
    FOR x:=2 TO start DO PRINT q(x),
    PRINT "(", // «9»
    FOR x:=start+1 TO digit DO PRINT q(x),
    PRINT ")",
  ELSE
    FOR x:=2 TO digit DO PRINT q(x),
  ENDIF
  PRINT
  PRINT "enter numerator, denominator"
  INPUT "enter 0,0 to quit: ": n,d
  n:=INT(ABS(n)); d:=INT(ABS(d))
ENDWHILE
```

These are the explanations for our notes in the program listed above:

«1» *variable names are shortened to avoid wrap lines. We usually don't do this, since long variable names are desirable.*

«2» *same as «1»*

«3» *checks for the denominator equal to zero to avoid a division by zero error later.*

«4» *the index into the two arrays q and r are now in sync.*

«5» *multiplication by 10 is done differently*

«6» *this is the trick loop mentioned in our letter.*

«7» *this is added information that explains if the number includes repeating digits.*

«8» *this is the start of one unified print section.*

«9» *if start=1 then this loop is skipped. This is when there are no non-repeating digits.*

We appreciated the letter that Bill Inhelder sent to us in reply to our revised program. It caused us to look closer at the program. In doing so, we found that the program could be redone in a more readable form (listed at the end of this article). Here is his reply to our letter:

Dear Editor - Thank you for the demonstration that *decimal'rep* could be written without the GOTO statement. Your rewrite also demonstrated how a program could be redesigned more succinctly and refined with less redundancy.

May I share with you a concern that I have with the policy of not releasing any program with a GOTO statement. With all the drawbacks of GOTO in a structured modular language, it is nevertheless included in the accepted COMAL instruction set. I believe it has yet to be proven that *all* programs, especially ones containing complex algorithms, can be written in a straight forward, readable manner without the occasional use of a GOTO statement. This policy appears to me to be rather arbitrary and narrowly pedantic. It makes no sense to exclude programs which contain GOTOs, especially if the program is more readable as a result.

Let me cite *decimal'rep* and its accompanying article as a case in point. The first objective of the program and article was to present to programmers an algorithm for carrying out a task. The objective of the program was to make each stage of the algorithm and each case within the algorithm as clear and as readable as possible. This was done so that the reader could follow it easily, understand what was done, and then do exactly what you did. That is, improve it by making it more succinct, condensed, less redundant, but not necessarily more readable. Programming subtleties (ie. not executing a FOR loop because the upper bound exceeds the lower bound) however elegant do not lead to readability and understanding. In any written language redundancy is necessary for fuller understanding. Thus a reworked, finely tuned program may be more efficient and have an appeal to elegance, but it does not necessarily make it more understandable. The natural division of the algorithm into its various cases was done deliberately so that the reader would comprehend each stage. The GOTOs were

more»

included at each stage so that the user would not have to rerun the program for another example.

The second objective of the article and program was directed at the user who would apply the program as a tool for a mathematical investigation. Your improved program serves this objective excellently; however, I feel that it would be self-defeating in terms of the original objective.

I would hope that an objective of this magazine is to improve programming skills. What better way to help to accomplish this objective than to include programs and articles which challenge the reader to improve on the program. Why not include some programs with GOTOs to see if your readers can rewrite them without GOTOs and at the same time make them more readable.

It was not my intention in writing this letter to belittle or show lack of appreciation for the obvious improvement over the original program. My concern is solely with the limitations imposed by the rigidity of the policy. The bottom line ought to be the degree to which the program is readable and understandable, and not whether it contains a GOTO statement.
Sincerely, Bill Inhelder

[We do not wish to print program listings showing how not to program (ie, with GOTO statements). Rather we fix the program first. Another way to modify his original program is shown below. It maintains long variable names and adds procedures and functions. This makes it easier to read the main program, but has more wrap lines than usual. It should work with all COMAL 2.0 systems unchanged.]

```
max'digits:=100 // decimal'rep'fin
DIM quotient(0:max'digits)
DIM remainder(0:max'digits)
instructions
LOOP
  PRINT // blank line
  INPUT "numerator, denominator: ":
  numerator,denominator // wrap line
  numerator:=INT(ABS(numerator)+.5)
  denominator:=INT(ABS(denominator)+.5)
  EXIT WHEN denominator=0
  present:=0; original'numerator:=numerator
  LOOP
    quotient(present):=numerator DIV
    denominator // wrap line
    remainder(present):=numerator MOD
    denominator // wrap line
    EXIT WHEN exact OR repeating OR give'up
    numerator:=remainder(present)*10
    present:+1
  ENDLOOP
  print'result
ENDLOOP
//
PROC instructions
  PAGE
  PRINT "This program will output the decimal"
  PRINT "expansion of all fractions which"
  PRINT "repeat or terminate in less than 100"
  PRINT "digits. Because of the precision of"
  PRINT "the computer, the numerator and"
  PRINT "denominator should be less than 10"
  PRINT "digits. Repeating portion of decimal"
  PRINT "expansion is enclosed in parentheses."
  PRINT "To Quit enter  0,0"
ENDPROC instructions
//
FUNC exact
  RETURN remainder(present)=0
ENDFUNC exact
//
FUNC repeating
  FOR past:=0 TO present-1 DO
    IF remainder(past)=remainder(present) THEN
    RETURN past+1 // repeating // wrap line
  ENDFOR past
  RETURN FALSE // not repeating yet
ENDFUNC repeating
//
FUNC give'up
```

more»

```
  IF present=max'digits THEN
    RETURN TRUE // give up
  ELSE
    RETURN FALSE // continue
  ENDIF
ENDFUNC give'up
//
PROC print'result
  PRINT original'numerator,"/",denominator;"=";
  PRINT quotient(0),".", // integer part
  IF exact THEN
    FOR digit:=1 TO present DO PRINT
    quotient(digit), // wrap line
    PRINT " exactly"
  ELIF repeating THEN
    FOR digit:=1 TO repeating-1 DO PRINT
    quotient(digit), // non-repeating part//wrap
    PRINT "(",
    FOR digit:=repeating TO present DO PRINT
    quotient(digit), // repeating part// wrap
    PRINT ") (repeats)"
  ELSE // does not repeat
    FOR digit:=1 TO max'digits DO PRINT
    quotient(digit), // wrap line
    PRINT " approximately"
  ENDIF
ENDPROC print'result
```

Here are examples of the final in action:

```
numerator, denominator: 1,7
1/7 = 0.(142857) (repeats)

numerator, denominator: 1,6
1/6 = 0.1(6) (repeats)

numerator, denominator: 17,64
17/64 = 0.265625 exactly

numerator, denominator: 101,56
101/56 = 1.803(571428) (repeats)

numerator, denominator: 0,0
«program ends»
```

Best Books

The good news is that *COMAL Collage* is now in stock and that three more new books should be ready in a couple weeks: *Packages Library Vol 2*, *COMAL Cross Reference*, and *Three Programs in Detail*. However our other two book projects (*Today Tutorials* and *Today Tips and Notes*) are way behind schedule and are delayed until September. Here are the charts of the best selling books for April, May and June:

April 1987

#1 - COMAL From A to Z
by Borge Christensen
#2 - COMAL Workbook
by Gordon Shigley
#3 - COMAL 2.0 Packages
by Jesse Knight
#4 - Packages Library #1
by David Stidolph
#5 - COMAL Today - The Index
by Kevin Quiggle

May 1987

#1 - Graph Paper
by Garrett Hughes
#2 - COMAL From A to Z
by Borge Christensen
#3 - Cartridge Graphics & Sound
by Captain COMAL's Friends
#4 - COMAL Today - The Index
by Kevin Quiggle
#5 - COMAL Handbook
by Len Lindsay

June 1987

#1 - COMAL Workbook
by Gordon Shigley
#2 - Cartridge Graphics & Sound
by Captain COMAL's Friends
#3 - Graph Paper
by Garrett Hughes
#4 - COMAL Handbook
by Len Lindsay
#5 - COMAL Collage
by Frank & Melody Tymon ■

Reversi

by Captain COMAL 

Reversi, a two player strategy game on *Today Disk #18*, is played on a checker board. There are 64 two color game pieces. One side is black, the other white. Diagram 1 shows the starting board using X and o to represent the two players.

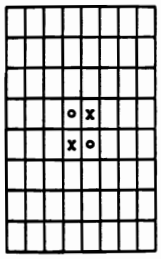


Diagram 1

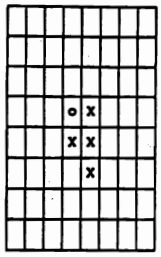


Diagram 2

The players alternate turns, placing one new marker of their color on the board each turn. Each new marker placed must capture at least one of the other players pieces. To capture a piece, trap it between your new marker and one of your old ones. You can trap a piece between two of your pieces on the same row, column, or diagonal. Diagram 2 shows the board after the X player moves. If you can't place a marker during your turn you lose, otherwise the player with the most pieces at the end is the winner.

You may move first or second when playing against the computer, or the computer can play against itself. During game play, a row of X or o is displayed below the game board indicating whose turn it is. When it is your turn, a star (*) will be placed on the screen to suggest a possible move. You may press «*return*» to accept the move or select another square to place your new marker on. The screen will remind you that the I, J, K, and L keys are used to move the cursor.

Once you make your selection, the computer flips the captured pieces. After thinking about it's move, the computer will place its own new marker, and flip the markers it captures. The following two screen dumps are taken from the middle of a game. The first has a star (*) at the location X is about to move to. The other shows the result of move. Notice the markers which were flipped from o to X.

Diagram 3

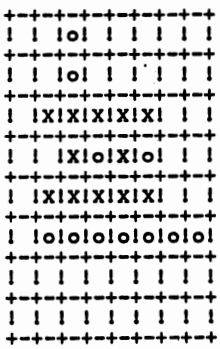


Diagram 4

The computer plays well, but is not an expert. While it will challenge a beginner, an experienced player should be able to win.

This leads to a new programming challenge. The program consists of several modules. The main routine allows each player to take a turn. Supporting routines handle tasks such as setting up the game board, determining if a move is allowed, flipping the pieces, and keeping score. These routines should not have to be changed. The strategy procedure suggests a valid move for the player and makes the choice for the computer. This is the procedure which needs to be altered to improve the computer's strategy. Feel free to alter this procedure, and watch the results.

Special program note: as listed, the program is designed for one player against the computer and will work with all COMAL 2.0 systems. To use this game with two people, just replace the fourth line with the line below it. Remember to remove the comment (//) symbols at the start of the line.

more»

```
init
questions
REPEAT
  IF player=person THEN choose'square(
  x,y,board#(,),piece$()) // wrap line
  //choose'square(x,y,board#(,),piece$())//2people
  IF can'flip(x,y,player,board#(,)) THEN
    flip(x,y,player,score(),board#(,),piece$(
    player)) // wrap line
    IF score(1)+score(2)<64 THEN
      player:=player MOD 2+1
      scoreboard
      strategy(x,y,player,board#(,),game'over)
      IF NOT game'over THEN flash(x,y,"*")
    ELSE
      game'over:=TRUE
    ENDIF
  ELSE
    INPUT AT 23,1,1: "press return to try
    again": ret$ // wrap line
    PRINT AT 23,1: SPC$(39),
  ENDIF
UNTIL game'over
show'winner
//
PROC init
  DIM board#(8,8), score(2)
  DIM piece$(0:2) OF 1
  player:=1; game'over:=FALSE
  score(1):=2; score(2):=2
  x:=5; y:=3; piece$(0):=" "
  piece$(1):="X"; piece$(2):="o"
  draw'board
  board#(4,4):=1; board#(4,5):=2
  board#(5,4):=2; board#(5,5):=1
  place'piece(4,4,piece$(1))
  place'piece(4,5,piece$(2))
  place'piece(5,4,piece$(2))
  place'piece(5,5,piece$(1))
  scoreboard
ENDPROC init
//
PROC questions
  PRINT AT 22,1: "enter 0) for a demo game"
  TRAP
    INPUT AT 23,1,1: "1) to go first, 2) to go
    last: ": ret$, // wrap line
    person:=VAL(ret$)
  HANDLER
    person:=0
  ENDTRAP
  PRINT AT 22,1: SPC$(39),
  PRINT AT 23,1: SPC$(39),
  IF person<0 OR person>2 THEN person:=0
ENDPROC questions
//
PROC scoreboard
  PRINT AT 9,2: USING piece$(1)+"=##":
  score(1) // wrap line
  PRINT AT 9,35: USING piece$(2)+"=##":
  score(2) // wrap line
  PRINT AT 19,12: piece$(player),
  FOR x:=1 TO 16 DO PRINT piece$(player),
ENDPROC scoreboard
//
PROC draw'board
  PAGE
  PRINT TAB(12),  "+-+-+-+-+-+-+-+-+"
  FOR x:=1 TO 7 DO
    PRINT TAB(12),"! ! ! ! ! ! ! ! !"
    PRINT TAB(12),"+-+-+-+-+-+-+-+-+"
  ENDFOR x
  PRINT TAB(12),  "! ! ! ! ! ! ! ! !"
  PRINT TAB(12),  "+-+-+-+-+-+-+-+-+"
ENDPROC draw'board
//
PROC place'piece(x,y,piece$)
  PRINT AT 18-2*y,11+2*x: piece$,
ENDPROC place'piece
//
PROC flash(x,y,token$)
  FOR f:=1 TO 5 DO
    place'piece(x,y,token$)
    FOR pause:=1 TO 100 DO NULL
    place'piece(x,y," ")
    FOR pause:=1 TO 100 DO NULL
  ENDFOR f
ENDPROC flash
//
PROC choose'square(REF x,REF y,REF board#(,),
piece$()) CLOSED // wrap line
  IMPORT place'piece
```

more»

```
  DIM choice$ OF 1
  WHILE KEY$>CHR$(0) DO NULL
  PRINT AT 21,1: "These keys move the *:   I"
  PRINT AT 22,24: "J K L"
  PRINT AT 24,1: "Hit <return> to choose the *
  square", // wrap line
  REPEAT
    IF board#(x,y) THEN
      place'piece(x,y," ")
    ELSE
      place'piece(x,y,"*")
    ENDIF
    choice$:=KEY$
    place'piece(x,y,piece$(board#(x,y)))
    CASE choice$ OF
    WHEN "i","I" // up
      y:=y MOD 8+1
    WHEN "l","L" // right
      x:=x MOD 8+1
    WHEN "k","K" // down
      y:=(y-1)+8*(y=1)
    WHEN "j","J" // left
      x:=(x-1)+8*(x=1)
    OTHERWISE
    ENDCASE
  UNTIL choice$=CHR$(13)
  PRINT AT 21,1: SPC$(39)
  PRINT SPC$(39)
  PRINT AT 24,1: SPC$(39),
ENDPROC choose'square
//
PROC check'dir(x0,y0,player,REF board#(,),REF
flipped()) CLOSED // wrap line
  FOR d#:=1 TO 8 DO flipped(d#):=0
  IF board#(x0,y0) THEN RETURN
  other:=player MOD 2+1
  RESTORE
  DIM max(8)
  max(1):=7-y0; max(3):=7-x0
  max(5):=y0-2; max(7):=x0-2
  IF x0>y0 THEN
    max(2):=7-x0; max(6):=y0-2
  ELSE
    max(2):=7-y0; max(6):=x0-2
  ENDIF
  IF 9-x0>y0 THEN
    max(4):=y0-2; max(8):=x0-2
  ELSE
    max(4):=7-x0; max(8):=7-y0
  ENDIF
  FOR d#:=1 TO 8 DO
    READ dx,dy
    IF max(d#)>0 THEN
      x:=x0+dx; y:=y0+dy
      WHILE flipped(d#)<max(d#) AND board#(
      x,y)=other DO // wrap line
        flipped(d#):+1; x:+dx; y:+dy
      ENDWHILE
      IF board#(x,y)<>player THEN flipped(
      d#):=0 // wrap line
    ENDIF
  ENDFOR d#
  DATA 0,1,1,1,1,0,1,-1
  DATA 0,-1,-1,-1,-1,0,-1,1
ENDPROC check'dir
//
FUNC can'flip(x,y,player,REF board#(,)) CLOSED
  IMPORT check'dir
  DIM directions(8)
  check'dir(x,y,player,board#(,),directions())
  total:=0
  FOR d:=1 TO 8 DO total:+directions(d)
  RETURN total
ENDFUNC can'flip
//
PROC flip(x,y,play,REF score(),REF board#(,)
,token$) CLOSED // wrap line
  IMPORT place'piece,check'dir
  DIM directions(8)
  RESTORE
  place'piece(x,y,token$)
  check'dir(x,y,play,board#(,),directions())
  board#(x,y):=play; score(play):+1
  other:=play MOD 2+1
  FOR d:=1 TO 8 DO
    READ dx,dy
    FOR t:=1 TO directions(d) DO
      board#(x+dx*t,y+dy*t):=play
      place'piece(x+dx*t,y+dy*t,token$)
      score(play):+1; score(other):-1
    ENDFOR t
  ENDFOR d
```

more»

```
  DATA 0,1,1,1,1,0,1,-1,0,-1,-1,-1,-1,0,-1,1
ENDPROC flip
//
PROC strategy(REF x,REF y,player,REF board#(
,),REF game'over) CLOSED // wrap line
  IMPORT can'flip,check'dir
  PRINT AT 23,1: "thinking",
  DIM squares(8,2), mix(8), order(8,0:1)
  RANDOMIZE
  FOR s:=1 TO 8 DO mix(s):=s
  FOR s:=1 TO 8 DO
    t:=RND(s,8); order(s,1):=mix(t); mix(t):=mix(s)
  ENDFOR s
  FOR s:=1 TO 4 DO mix(s):=s
  FOR s:=1 TO 4 DO
    t:=RND(s,4); order(s,0):=mix(t); mix(t):=mix(s)
  ENDFOR s
  RESTORE
  WHILE NOT EOD DO
    READ count
    FOR s:=1 TO count DO READ squares(s,1)
    ,squares(s,2) // wrap line
    FOR s:=1 TO count DO
      r:=squares(order(s,count=8),1)
      c:=squares(order(s,count=8),2)
      IF NOT board#(r,c) THEN
        IF can'flip(r,c,player,board#(,)) THEN
          x:=r; y:=c
          PRINT AT 23,1: SPC$(8),
          RETURN
        ENDIF
      ENDIF
    ENDFOR s
  ENDWHILE
  game'over:=TRUE
  DATA 4,1,8,8,8,8,1,1,1,4,6,3,6,6,3,6,3,3
  DATA 8,5,6,4,6,3,5,3,4,4,3,5,3,6,4,6,5
  DATA 8,3,8,6,8,8,6,8,3,6,1,3,1,1,3,1,6
  DATA 8,6,7,7,6,7,3,6,2,3,2,2,3,2,6,3,7
  DATA 8,1,5,1,4,4,1,5,1,8,4,8,5,5,8,4,8
  DATA 8,4,7,5,7,7,5,7,4,5,2,4,2,2,4,2,5
  DATA 8,1,2,2,1,7,1,8,2,8,7,7,8,2,8,1,7
  DATA 4,2,7,7,7,7,2,2,2
ENDPROC strategy
//
PROC show'winner
  PRINT AT 9,2: USING piece$(1)+"=##":
  score(1) // wrap line
  PRINT AT 9,35: USING piece$(2)+"=##":
  score(2) // wrap line
  IF score(1)+score(2)=64 THEN
    IF score(1)>score(2) THEN
      PRINT AT 21,1: "player "+piece$(1)+" won"
    ELIF score(1)=score(2) THEN
      PRINT AT 21,1: "tie"
    ELSE
      PRINT AT 21,1: "player "+piece$(2)+" won"
    ENDIF
  ELSE
    PRINT AT 21,1: "player "+piece$(player MOD
    2+1)+" won", // wrap line
  ENDIF
ENDPROC show'winner ■
```


JOIN THE ON-LINE COMMODORE® USER GROUP.

Imagine being part of a nationwide on-line user group. With new QuantumLink, you can instantly exchange ideas, information and software with Commodore users everywhere, and participate in live discussions with Commodore experts.

And you can participate in conferences held by Len Lindsay, access COMAL public domain programs, and have your questions answered by other Comalites. You can even share your public domain COMAL programs with others.

These are just a few of the hundreds of features available. If you already have a modem, you can register on-line for a free software kit and trial subscription. Hook up and call **800-833-9400**. If you need a modem, call QuantumLink Customer Service at 800-392-8200.

QuantumLink™
The Commodore® Connection

Poetry 

by Dick Klingens, Dutch COMAL Users Group

A programming language can be described by production rules. Such rules for COMAL are called the COMAL Kernal and can be found on page 40 of *COMAL Today #17*. A context free grammar is a formal description of the grammar of a language. Notions of the language are described (or defined) in terms of other notions; finally in terms of words from the language. The production rules for COMAL form an almost context free grammar. Included are:

«identifier» means: the notion *identifier* or, *what is an identifier*. Notions not written inside « » are called terminal symbols.
::= means: is defined by.
| means: OR.

For example, the «integer» rules might be:

1. «integer»::=«digit» | «digit»«integer»
2. «digit»::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

To find out whether 1234 is an «integer» or not we have:

1234 = 1 234

Following the second part of rule 1: 1 is a digit. Next we examine 234. Is it an integer?

234 = 2 34

2 is a digit. Is 34 an integer?

34 = 3 4

3 is a digit. Now, is 4 an integer? Yes, that follows from first part of rule 1.

Therefor, 1234 is an integer.

The COMAL Kernal describes in a formal way what a program is. The first production rule of the Kernal is:

«comal program»::=«block»

Parts of a natural language sometimes can be described in the same way. There is, however, a slight (big) difference: sentences are sometimes meaningless but syntactically correct according to the rules.

A Context Free Grammar

In the program *grammar.lst* (for 0.14 and 2.0 on *Today Disk #18*, 0.14 side), I describe a context free grammar for some English sentences, inspired by the song *Mary had a Little Lamb.*

To produce unexpected sentences, I used several syntax constructions like:

«article»::=[0.45] a | the

which means the probability for choosing "a" as an «article» is 0.45; the alternative "the" has probability 1 - 0.45 = 0.55. "A" and "the" are terminal symbols.

The production of a sentence starts with
«sentence»::=«subject»«predicate» «endsymbol»

«subject»::=[0.7]«substantive group» |
«personal pronoun»
«substantive group»::=«article»«noun phrase»

It is easy transform the grammar into a COMAL program that produces the sentences. Here is part of the program. This will give you a general idea of the production rules:

```
PROC sentence
  subject; predicate; endsymbol
ENDPROC sentence
//
PROC subject
  IF p(0.7) THEN
    substantive'group
  ELSE
    personal'pronoun
```

more»

```
  ENDIF
ENDPROC subject
//
PROC substantive'group
  article; noun'phrase
ENDPROC substantive'group
//
PROC article
  IF p(0.45) THEN
    symbol("a")
  ELSE
    symbol("the")
  ENDIF
ENDPROC article
//
PROC endsymbol
  PRINT
ENDPROC endsymbol
//
FUNC p(x) CLOSED
  RETURN RND<x
ENDFUNC p
```

To print terminal symbols on the screen we use:

```
PROC symbol(x$)
  PRINT x$+" ",
ENDPROC symbol
```

The grammar is called a probablistic context free language because probabilities are used during production. Some sentences produced are:

- the bear that sees the bear likes a bad bear
- the bear likes him
- the boy dreams about the bear
- a girl never sees the girl
- Jim sees Mary
- a boy that sees the big tree always likes a boy that always eats a tree
- Mary eats a little bear

In context free grammars recursion is often used to define notions:

«noun phrase»::=
  [0.25]«adjective»«noun phrase» | «nounpart»

And because COMAL itself allows recursion, transforming this rule can be accomplished in a very natural way.

```
PROC noun'phrase
  IF p(0.25) THEN
    adjective
    noun'phrase // COMAL recursion!
  ELSE
    nounpart
  ENDIF
ENDPROC noun'phrase
```

HAIKU

Today Disk #18 also includes a program called *haiku* (a type of poetry). The haiku originated in Japan about four centuries ago. In its most classical version the haiku consists of three lines, with five syllables in the first line, seven syllables in the second and five in the third.

The haiku-composing program has a built-in vocabulary of 124 words. The production of the poems is described by production rules of the same kind that was described above. However, because the Japanese syllable-sound is not equivalent to the English syllable, poems created are not limited to the 5-7-5 pattern.

One of the four haiku patterns which are used by the program, is:

«article»«adjective»«noun»...
«article»«noun»«verb»«preposition»«article»
    «noun»
«adjective»«adjective»«noun»

generating a haiku such as:

A spring meadow...
the moon sleeps under a bush
broken blue snowflake ■

Hammurabi 

by Mark Skopinsky

In this game you play the part of Hammurabi, the ruler of the ancient city-state of Sumeria, for a ten-year term of office. The city starts with 1,000 acres of land, a population of 100 people, and 3,000 bushels of grain in the city storehouses. You may buy and sell land with neighboring city-states for grain. The price will fluctuate between 17 and 26 bushels per acre. You must also use the grain to feed the people and for seed to plant crops each year.

You will find that there are many things involved in running the city. People can only work a certain amount of land, and they will starve if not fed enough. You also have to deal with things such as rats eating the harvested grain, plagues, and poor harvests. Ruling a kingdom is no easy task! *Hamurabi.lst* works on all versions of COMAL and is LISTed to disk on *Today Disk #18*, 0.14 side.

```
// Mytech users - replace RND with RND#
init
FOR year:=1 TO 10 DO
  begging'steward
  report'update
  buy'land
  IF buy=0 THEN sell'land
  feed'people
  plant'crops
  do'rats'attack
  pop'increase
  plague'chance
ENDFOR year
final'report
end'game
//
PROC init
  pcnt'dead:=0;pop'pcnt:=0
  pop:=95;grain:=3;acres:=1000
  harvest:=3000;rat'food:=200
  stored:=harvest-rat'food
  settlers:=5;num'dead:=0
ENDPROC init
PROC begging'steward
  FOR x:=1 TO 25 DO PRINT // clear screen
  PRINT "Hammurabi, your humble steward";
  PRINT "begs to"
  PRINT "report that in Year";
  PRINT year;"of your reign:"
  PRINT
  IF num'dead=0 THEN
    PRINT "None of your loyal";
    PRINT "subjects perished,"
  ELSE
    PRINT num'dead;
    PRINT "of your loyal subjects perished,"
  ENDIF
  PRINT "and";settlers;
  PRINT "newcomers settled in the city."
  PRINT
  pop:+settlers
ENDPROC begging'steward
//
PROC report'update
  PRINT "Your subjects now number";pop,"."
  PRINT
  PRINT "The city owns";acres;"acres of land."
  PRINT "The harvest yielded";harvest;"bushels,"
  PRINT USING "or #.# bushels harvested per
  acre":harvest/acres// wrap line
  IF rat'food=0 THEN
    PRINT "Rodents ate none of the grain."
  ELSE
    PRINT "Rodents ate";rat'food;
    PRINT "bushels of grain."
  ENDIF
  PRINT "The granaries now hold";stored;
  PRINT "bushels."
  PRINT
  price:=RND(17,26)
  PRINT "Land is trading at";price;
  PRINT "bushels per acre."
ENDPROC report'update
//
PROC buy'land
  REPEAT
    INPUT "How many acres do you desire?
    ":buy// wrap line
    IF price*buy>stored THEN wrong'grain
```

more»

```
  UNTIL buy*price<=stored
  acres:+buy;stored:-grain*buy
ENDPROC buy'land
//
PROC sell'land
  REPEAT
    INPUT "How many acres should I sell?
    ":sell// wrap line
    IF sell>acres THEN wrong'acres
  UNTIL sell<=acres
  acres:-sell;stored:+price*sell
ENDPROC sell'land
//
PROC feed'people
  REPEAT
    PRINT
    PRINT "How many bushels of grain do"
    INPUT "you wish to feed your people?
    ":food// wrap line
    IF food=0 THEN steward'quits
    IF food>stored THEN wrong'grain
  UNTIL food<=stored
  stored:-food
ENDPROC feed'people
//
PROC plant'crops
  REPEAT
    ok:=FALSE
    PRINT
    PRINT "How many acres of land do"
    INPUT "you wish to plant with grain?
    ":plant// wrap line
    PRINT
    IF plant>acres THEN
      wrong'acres
    ELIF plant/2>stored THEN
      wrong'grain
    ELIF plant>10*pop THEN
      PRINT "But you have only";pop;"subjects"
      PRINT "to tend the fields! Now then,"
    ELSE
      ok:=TRUE
    ENDIF
  UNTIL ok
  stored:-(plant DIV 2)
ENDPROC plant'crops
//
PROC do'rats'attack
  production:=RND(1,6); rat'food:=0
  IF RND(1,100)>50 THEN
    rat'food:=stored DIV RND(1,6)
  ENDIF
  harvest:=stored+plant*production
  stored:=harvest-rat'food
ENDPROC do'rats'attack
//
PROC pop'increase
  num'dead:=0
  settlers:=INT(RND(1,6)*(20*acres+stored
  )/pop/100+1)// wrap line
  meals:=food DIV pop
ENDPROC pop'increase
//
PROC plague
  num'dead:=pop DIV 2
  pop:-num'dead
  PRINT "A horrible plague struck! Half of the"
  PRINT "people died."
ENDPROC plague
//
PROC plague'chance
  IF meals<20 THEN
    IF RND(1,100)<pcnt'dead THEN
      plague
    ELSE
      starve'the'population
    ENDIF
  ENDIF
ENDPROC plague'chance
//
PROC starve'the'population
  num'dead:=pop-INT(pop*(meals/20))
  IF num'dead>0.45*pop THEN starved
  pop'pcnt:=((year-1)*pop'pcnt+num'dead*100/pop
  )/year// wrap line
  pop:-num'dead
  pcnt'dead:+num'dead
ENDPROC starve'the'population
//
PROC starved
  PRINT
  PRINT "You starved";num'dead;
```

more»

```
  PRINT "people in one year!!"
  mismanagement
ENDPROC starved
//
PROC mismanagement
  PRINT "Due to this extreme";
  PRINT "mismanagement you"
  PRINT "have not only been impeached and"
  PRINT "thrown out of office, but you have"
  PRINT "also been declared National Fink!!!"
  end'game
ENDPROC mismanagement
//
PROC wrong'grain
  PRINT
  PRINT "Hammurabi, think again.";
  PRINT "You have only"
  PRINT stored;"bushels of grain. Now then,"
ENDPROC wrong'grain
//
PROC wrong'acres
  PRINT
  PRINT "Hammurabi, think again.";
  PRINT "You only own"
  PRINT acres;"acres. Now then,"
ENDPROC wrong'acres
//
PROC steward'quits
  PRINT
  PRINT "Hammurabi, I cannot do";
  PRINT "what you wish."
  PRINT "Get yourself another steward!!!!!!"
  end'game
ENDPROC steward'quits
//
PROC final'report
  PRINT "In your 10-year term of office,";
  PRINT INT(pop'pcnt),"%"
  PRINT "of the population starved per year,"
  PRINT "on the average, I.E. a total of";
  PRINT INT(pcnt'dead)
  PRINT "people died!!"
  land:=acres/pop
  PRINT
  PRINT "You started with 10 acres per person"
  PRINT "and ended with";land;"per person."
  PRINT
  FOR t:=1 TO 1000 DO NULL
  IF pop'pcnt>33 OR land<7 THEN
    mismanagement
  ELIF pop'pcnt>10 OR land<9 THEN
    nero'and'ivan
  ELIF pop'pcnt>3 OR land<10 THEN
    fair
  ELSE
    very'good
  ENDIF
ENDPROC final'report
//
PROC very'good
  PRINT "A fantastic performance!";
  PRINT "Charlemagne,"
  PRINT "Disraeli, and Jefferson";
  PRINT "combined could"
  PRINT "not have done better!"
ENDPROC very'good
//
PROC nero'and'ivan
  PRINT "Your heavy-handed performance";
  PRINT "smacks of"
  PRINT "Nero and Ivan the Terrible.";
  PRINT "The people"
  PRINT "(remaining) find you an unpleasant"
  PRINT "ruler, and, frankly, hate your guts!!"
ENDPROC nero'and'ivan
//
PROC fair
  PRINT "Your performance could";
  PRINT "have been some-"
  PRINT "better, but really wasn't too bad."
  PRINT RND(1,pop*0.8);
  PRINT "people would dearly like to see you"
  PRINT "assassinated, but we all have our"
  PRINT "trivial little problems."
ENDPROC fair
//
PROC end'game
  PRINT
  PRINT "So long for now."
  PRINT
  END // of the game
ENDPROC end'game ■
```

First COMAL 0.14 Program

by Gary Franklin 

COMAL is an advanced programming language which can be used for powerful applications or as a stepping stone to other professional languages. However, it is even better for the absolute computer beginner. The power of a language may not be important to you if you can't write your first program in it. This demonstration will ease you into the world of programming.

COMAL is not built into your c64. You must first load it into your computer from a disk. If you have a disk with COMAL, load it into your computer before continuing with this article.

Now that COMAL is loaded into your computer, type the «*shift*» and «*clr home*» keys together to clear your screen. Then type:

new «*return*»

This erases any program in memory. Next type:

setgraphic 0 «*return*»

If you made a mistake typing this, COMAL will tell you, as you may already have found out. A common problem in typing commands in COMAL is forgetting to put a space after words (between the setgraphic and the digit zero in this case). Also be sure to press the «*return*» key after each line. (I won't include the reminder anymore.)

If all went well, you will see a split screen. The top 2 lines are reserved for text; there is even a blinking cursor. The rest of the screen is blank except for a small triangle. The triangle is called a turtle.

The function keys, **f1** and **f3**, now have a special meanings. Press **f1** to return to the text screen, then **f3** to go back to the split screen. There is no turtle on the text screen and it is possible for the turtle to disappear from the split screen. If this happens, type these commands to get it back:

showturtle
home

Go back to the split screen (use the **f3** key). Then type:

forward 50

If all went well, the turtle will climb towards to top of the screen. It will leave a trail behind it. If you had a problem, COMAL returns you to the text screen and gives you an error message. Hit **f3** to go back to the split screen and try to type in the line again. Be careful how you type, and be sure to include a space after the word forward. Next type:

right 90

The turtle should turn 90 degrees to the right and face the right side of the screen. If you made a mistake, hit **f3** and try to enter the line more carefully.

If you haven't made any mistakes yet, congratulations. Press the **f1** key. You should see the commands that you have typed on the text screen. This may be helpful later. Press the **f3** key to go back to the split screen.

Now that you made it this far, the rest should be easy. Type:

forward 50
right 90
forward 50
right 90
forward 50

At this point, you should still be in split screen mode. There should be a square in the middle of the screen and the turtle should be facing

more»

towards the left. Press **f1** to go back to the text screen. If you were a perfect typist, everything you typed starting from NEW will still be on the screen. It will be very easy to make a second square. Move the cursor to the to the first line on the screen which has the **FORWARD 50** command, but don't press «*return*» yet.

Press **f3** to go back to the split screen mode. The top line of the screen should say **FORWARD 50** and have a blinking cursor. The square should still be in the middle of the screen. If this is the case, hit «*return*» 7 times. Now there is a second square to the left of the first. Wasn't that easy?

Now that you know how to draw a square, let's teach the computer to draw one for you. A program stores the instructions that teach the computer how to do things. Hit **f1** to go back to the text screen. If you are good with the cursor keys, you can insert line numbers before the commands that are already typed on the screen. If not, type in the following lines remembering to use unshifted characters, putting a space after each command word, and pressing the «*return*» key after each line:

```
0010 setgraphic 0
0020 forward 50
0030 right 90
0040 forward 50
0050 right 90
0060 forward 50
0070 right 90
0080 forward 50
```

If the computer tells you that you made a mistake, retype the line. When you are finished, type **RUN**. You will be sent back to the split screen, it will quickly be erased, and then the turtle will race around the screen, leaving a trail behind. Aren't you feeling good about yourself? You just wrote your first COMAL program and I bet you aren't even tired yet.

It is easy to give COMAL a program, but it is also easy to teach COMAL new words. Go back to the text screen (remember how). Now type:

```
15 proc square
90 endproc
run
```

What happened? The screen cleared, but the turtle didn't race around this time. However, COMAL has learned a new word. Type:

```
square
```

The turtle should draw a square. Type square three more times and you have four squares. To put the these squares into our program, go back to the text screen and type these lines:

```
11 for x=1 to 4 do square
12 right 45
13 for x=1 to 4 do square
run
```

Wasn't that pretty? Go to the text screen. Type a few more lines before we review:

```
renum
list
```

You should see:

```
0010 setgraphic 0
0020 for x:=1 to 4 do square
0030 right 45
0040 for x:=1 to 4 do square
0050 proc square
0060   forward 50
0070   right 90
0080   forward 50
0090   right 90
0100   forward 50
0110   right 90
0120   forward 50
0130 endproc square
```

more»

Now that the program is finished, you should save it. It is a bad idea to save programs on original disks such as the *Today Disks* we distribute. If you have a disk with free blocks, use it. If you have a blank disk, you must format it first. To format a blank disk, insert it into the disk drive and type:

pass "n0:comal programs1,cp"

Now that you have a formatted disk in the drive, type:

save "first'program"

Later, when you want to run the program again, you must first load in COMAL. (Your program won't run from BASIC.) Then type:

load "first'program"
run

Wow, that was a lot for an hour or so. But now you can make your computer do things you may only have dreamed of before. You know that COMAL is a programming language. You must first load it into your c64, before you can use its commands. Once COMAL is loaded, you can use its commands directly from the keyboard (immediate mode) or you can put them together into a program. COMAL can even learn new words which you can use as easily as the built in commands. Finally, you learned how to save your program for later use.

Graphics are only a small part of COMAL. COMAL's powerful programming structures were only briefly touched on by using a FOR loop and a procedure. Even the **RENUM** command probably still seems like magic. You can't learn a programming language in one hour, but with COMAL, you can do something useful in that first hour. Happy programming. ■

Clock



by Gerard Busker, Dutch COMAL Users Group

Would you like the correct time in front of you on the screen? Not only during a program run, but also while editing? Impossible? No, **USE clock!**

LINK in the **clock** package with:

LINK "pkg.clock"

The time is set with the usual COMAL commands:

```
USE system
settime("12:13:14")
```

The following commands are possible:

showtime(TRUE): Prints time on screen.
timepos(x,y): Set clock position on the screen. The default screen position is (1,1).
timecolor(n): Color of the digits. -1 sets the color to the current text color.

The package is rommed. With the package procedure **setrommed(FALSE)**, it is possible to ram it so it can be saved along with your programs. The function **rommed** returns the status of the package (rommed or not).

Pkg.clock and a demo program, *demo/clock*, are on *Today Disk #18*. The program below shows the main package commands:

```
USE clock
USE system
settime("12:13:14")
PAGE
timepos(12,20)
timecolor(1)
showtime(TRUE)
```

Note: this package uses the cassette buffer and conflicts with Super Chip running on a C128. ■

Puzzle



by Christopher Laprise

This is a graphic version of Captain COMAL's Puzzle program published in *COMAL Today #14*. I'd dare say this version is a good answer to the challenge printed after the article. I know it's a little late, but that's what procrastination and cassette failures can do.

Despite suggestions in *COMAL Today #14*, the Sprite Maker routine in the issue wasn't used. Instead, I used the **font'to'sprite** routine from page 58 of that issue to make a set of large numbers from 1 to 9 and small numbers from 0 to 5 which can be combined to create the numbers 1 through 15 in the puzzle. The routine **boxdef** also creates a frame sprite to center each number in.

Suppose the program is waiting for us to move and we decide to move the number **5** to the right. First, the frame sprite and the **5** sprite are positioned over the screen image and turned on. Then, the screen area behind the sprites is erased and the sprites are moved to the new position with a **movesprite** command. Finally, the sprites are copied to the screen behind them (in their new position) with the **stampsprite** command and the sprites are shut off. To move a number like **12**, a small **1** is positioned in the frame and the small **2** placed beside it. All three sprites are then moved together to the next square.

When the program starts, there is a short delay while the sprites are initialized. After the reading the instructions, you may begin solving the puzzle. Use a joystick (in port 2) to move the pieces.

[*Editor's note: this program is the best solution to our game challenge from COMAL Today #14, page 59. We thank all those who responded and sent in solutions.*]

```
init
LOOP
  start
  solve
  message
ENDLOOP
//
PROC message
  textstyle(2,3,0,1)
  pencolor(6)
  plottext(xjust#+18,30,"WINNER !")
  plottext(0,0,"Hit Button")
  REPEAT
    joystick(2,dummy,button)
  UNTIL button
ENDPROC message
//
PROC init
  PAGE
  USE sprites
  FOR i:=0 TO 2 DO spritecolor(i,0)
  xjust#:=156; yjust#:=94
  USE joysticks
  USE turtle
  hideturtle
  graphicscreen(0)
  window(0,319,0,199)
  background(12); border(6)
  pencolor(0)
  dims
  make'sprites
ENDPROC init
//
PROC start
  move'count:=0; times:=0
  display'goal
  shuffle
  move'count:=0
  display'board
  textstyle(1,1,0,0)
  plottext(xjust#+18,2,"Use joystick 2")
  textstyle(1,2,0,0)
  plottext(xjust#+18,66,"MOVES:")
ENDPROC start
//
PROC solve
```

more»

```
  REPEAT
    get'move
    move
  UNTIL winner
ENDPROC solve
//
PROC display'board
  clear
  FOR rows:=1 TO 4 DO
    FOR cols:=1 TO 4 DO
      assign(rows,cols)
      FOR i:=0 TO 2 DO
        stampsprite(i)
        hidesprite(i)
      ENDFOR i
    ENDFOR cols
  ENDFOR rows
ENDPROC display'board
//
PROC dims
  DIM p(1:4,1:4), stick(0:8)
  DIM shape$ OF 64, hold$ OF 64
ENDPROC dims
//
FUNC winner
  counter:=0
  FOR rows:=1 TO 4 DO
    FOR cols:=1 TO 4 DO
      counter:=(counter+1) MOD 16
      IF p(rows,cols)<>counter THEN RETURN
      FALSE // wrap line
    ENDFOR cols
  ENDFOR rows
  RETURN TRUE
ENDFUNC winner
//
PROC display'goal
  counter:=0
  FOR rows:=1 TO 4 DO
    FOR cols:=1 TO 4 DO
      counter:=(counter+1) MOD 16
      p(rows,cols):=counter
    ENDFOR cols
  ENDFOR rows
  display'board
  textstyle(1,2,0,1)
  plottext(20,160,"This is the goal:")
ENDPROC display'goal
//
PROC shuffle
  RANDOMIZE
  textstyle(1,2,0,1)
  plottext(xjust#+18,66,"Shuffling ... Wait")
  FOR times:=1 TO 75 DO
    fake'move
    move
  ENDFOR times
ENDPROC shuffle
//
PROC fake'move
  init'stick
  REPEAT
    direction:=RND(1,7)
  UNTIL stick(direction)=TRUE
ENDPROC fake'move
//
PROC init'stick
  empty'square
  FOR x:=0 TO 8 DO stick(x):=FALSE
  FOR x:=1 TO 8 STEP 2 DO stick(x):=TRUE
  IF row=1 THEN stick(5):=FALSE
  IF row=4 THEN stick(1):=FALSE
  IF col=1 THEN stick(3):=FALSE
  IF col=4 THEN stick(7):=FALSE
ENDPROC init'stick
//
PROC empty'square
  FOR rows:=1 TO 4 DO
    FOR cols:=1 TO 4 DO
      IF p(rows,cols)=0 THEN
        row:=rows; col:=cols
        RETURN
      ENDIF
    ENDFOR cols
  ENDFOR rows
  STOP "error - no empty square found"
ENDPROC empty'square
//
PROC move
  CASE direction OF
  WHEN 1
    IF NOT times THEN move'tile(row+1,col)
```

more»

```
    p(row,col):=p(row+1,col)
    p(row+1,col):=0
  WHEN 3
    IF NOT times THEN move'tile(row,col-1)
    p(row,col):=p(row,col-1)
    p(row,col-1):=0
  WHEN 5
    IF NOT times THEN move'tile(row-1,col)
    p(row,col):=p(row-1,col)
    p(row-1,col):=0
  WHEN 7
    IF NOT times THEN move'tile(row,col+1)
    p(row,col):=p(row,col+1)
    p(row,col+1):=0
  OTHERWISE
    STOP "error - illegal move"
  ENDCASE
  move'count:+1
  IF NOT times THEN
    textstyle(2,2,0,0)
    plottext(xjust#+82,66,STR$(move'count))
  ENDIF
ENDPROC move
//
PROC get'move
  init'stick
  REPEAT
    joystick(2,direction,dummy)
  UNTIL stick(direction)=TRUE
ENDPROC get'move
//
PROC assign(rw,cl)
  FOR i:=0 TO 2 DO identify(i,0)
  IF p(rw,cl)>0 AND THEN p(rw,cl)<10 THEN
    identify(0,16)
    spritepos(0,cl*18+xjust#,(5-rw)*18+yjust#)
    showsprite(0)
    identify(1,p(rw,cl))
    spritepos(1,cl*18+xjust#,(5-rw)*18+yjust#)
    showsprite(1)
  ELIF p(rw,cl)>9 AND THEN p(rw,cl)<16 THEN
    identify(0,16)
    spritepos(0,cl*18+xjust#,(5-rw)*18+yjust#)
    showsprite(0)
    identify(1,11); identify(2,p(rw,cl))
    spritepos(1,cl*18+xjust#,(5-rw)*18+yjust#)
    showsprite(1)
    spritepos(2,cl*18+8+xjust#,(5-rw)*18+yjust#)
    showsprite(2)
  ENDIF
ENDPROC assign
//
PROC erase(rw,cl)
  viewport(cl*18+xjust#,cl*18+xjust#+17,(5-rw)
  *18+yjust#-17,(5-rw)*18+yjust#) // wrap line
  clear
  viewport(0,319,0,199)
ENDPROC erase
//
PROC move'tile(y,x)
  assign(y,x); erase(y,x)
  FOR i:=0 TO 1 DO movesprite(i,col*18+xjust#,
  (5-row)*18+yjust#,10,0) // wrap line
  movesprite(2,col*18+8+xjust#,(5-row)*18
  +yjust#,10,0) // wrap line
  startsprites
  WHILE moving(0) OR moving(1) OR moving(2)
  DO NULL // wrap line
  FOR i:=0 TO 2 DO stampsprite(i); hidesprite(i)
ENDPROC move'tile
//
PROC make'sprites
  textstyle(1,2,0,0)
  plottext(70,66,"INITIALIZING: Please Wait")
  boxdef
  FOR i:=1 TO 9 DO
    shape$:=""
    font'spr(3,48+i,shape$,2,2)
    define(i,shape$)
    disp
  ENDFOR i
  shape$:=""
  font'spr(3,32,shape$,2,2)
  define(0,shape$)
  FOR i:=10 TO 15 DO
    shape$:=""
    font'spr(3,38+i,shape$,1,2)
    define(i,shape$)
    disp
  ENDFOR i
ENDPROC make'sprites
//
```

more»

```
PROC font'spr(fontset,char,REF sprite$,xsize
,ysize) CLOSED // wrap line
  DIM letter$ OF 8
  IF fontset<0 OR fontset>3 THEN REPORT 5
  IF char<0 OR char>255 THEN REPORT 5
  IF xsize<1 OR xsize>2 THEN REPORT 5
  IF ysize<1 OR ysize>2 THEN REPORT 5
  //error 5 is value out of range
  USE font
  getcharacter(fontset,char,letter$)
  sprite$:=""
  FOR row#:=1 TO 8 DO
    number:=0
    FOR bit'col#:=0 TO 7 DO
      bit#:=SGN((2^(7-bit'col#)) BITAND ORD(
      letter$(row#))) // wrap line
      FOR count#:=1 TO xsize DO number:=
      number*2+bit# // wrap line
    ENDFOR bit'col#
    number:=number*256^(3-xsize)
    FOR c#:=1 TO ysize DO
      sprite$:+CHR$(number DIV 65536)
      sprite$:+CHR$((number DIV 256) MOD 256)
      sprite$:+CHR$(number MOD 256)
    ENDFOR c#
  ENDFOR row#
  FOR count#:=1 TO 40 DO sprite$:+CHR$(0)
  sprite$:=sprite$(1:64)
  IF LEN(sprite$)<>64 THEN REPORT 4
  // error 4 is substring error
ENDPROC font'spr
//
PROC disp
  identify(1,i)
  spritepos(1,50,50)
  spritecolor(1,0)
  showsprite(1)
ENDPROC disp
//
PROC boxdef
  shape$:=CHR$(255)+CHR$(255)+""128""
  FOR i:=1 TO 16 DO shape$:+""128""0""128""
  shape$:+CHR$(255)+CHR$(255)+""128""
  FOR i:=1 TO 10 DO shape$:+CHR$(0)
  define(16,shape$); identify(0,16)
ENDPROC boxdef ■
```

Epicycloids



by Jack Baldrige

In 1985, a fascinating book, *Microcomputer Art*, was published. Its opening sentence explains one feature which makes it unique. It is described as perhaps the only work written about computers whose bibliography contains **no reference less than one hundred years old.**

The book shows how a microcomputer may be used to simulate the operation of a **geometric pen.** These devices were used to produce complex geometric patterns on wood, metal or glass. The patterns were epicycloids, defined as the path of a pen on the circumference of a secondary wheel rolling around a primary or fixed wheel. The book shows more than 600 such patterns, along with equations and parameters for duplicating them on a computer.

The author also shows how patterns are changed when the relationships between parameters are changed. The program *epicycloids* on *Today Disk #18* summarizes the results when parameters are varied in simple patterns, and shows a few of the more complex patterns given in the book. Ideally, a printer-plotter should be used to capture the detail of such patterns, but the program gives an idea, at least, of the variety of designs which may be obtained.

Reference: *Microcomputer Art*, by Ross Edwards Prentice-Hall of Australia Pty Ltd, 1985 ■

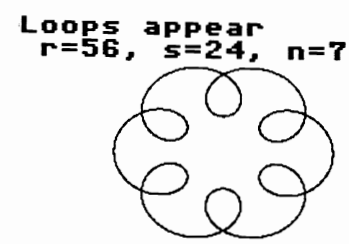


Figure 18

ZIP Zone



by Len Lindsay

COMAL is proving to be quite an all purpose language. In the past four years we have received about 5,000 COMAL programs from our readers. The diversity is apparent just looking at *COMAL Today*.

We use COMAL to run our order processing and inventory control systems here. Part of the order processing program is an automated zone calculator. Packages we ship via UPS must be clearly labeled and include the package weight and zone number. Finding the weight is easy. And UPS provides a chart that gives us the correct zone (based on distance from your local UPS office to the destination of the package). The zone chart will vary depending on your location. The chart below is for Madison. Ask your local UPS driver for your zone chart.

ZIP CODE PREFIXES	UPS ZONE	ZIP CODE PREFIXES	UPS ZONE	ZIP CODE PREFIXES	UPS ZONE	ZIP CODE PREFIXES	UPS ZONE
010-043	5	437-457	4	594-599	6	829-853	6
044	6	458	3			854	7
045	5	459	4	600-613	2	855-863	6
046-047	6	460-462	3	614-628	3	864	7
048-089	5	463-464	2	629	4	865-874	6
		465-469	3	630-635	3	875-877	5
100-139	5	470-472	4	636-676	4	878-880	6
140-147	4	473-474	3	677-679	5	881	5
148-149	5	475-477	4	680-689	4	882-883	6
150-168	4	478-499	3	690	5	884	5
169-199	5			691-692	4	890-892	7
		500-505	3	693-699	5	893	6
200-214	5	506-507	2			894-897	7
215	4	508	3	700-719	5	898	6
216-241	5	510-512	4	720-729	4		
242-243	4	513-514	3	730-739	5	900-961	7
244-245	5	515-516	4	740-744	4	970-978	7
246-253	4	520-524	2	745	5	979	6
254	5	525-526	3	746	4	980-986	7
255-268	4	527-539	2	747-748	5	988-989	7
270-299	5	540	3	749	4	990-992	6
		541-544	2	750-768	5	993	7
300-306	5	545	3	769	6	994	6
307	4	546-547	2	770-778	5		
308-326	5	548	3	779-785	6		
327-339	6	549	2	786-787	5		
342	6	550-558	3	788	6		
350-355	5	559	2	789-796	5		
356-358	4	560-561	3	797-799	6		
359-369	5	562	4				
370-386	4	563	3	800-812	5		
387-397	5	564-576	4	813-815	6		
		577	5	816-820	5		
400-433	4	580-585	4	821	6		
434-436	3	586-593	5	822-828	5		

The zone is determined by the first 3 digits of the destination zip code. That gives 999 possible zones (there is no 000). I use a 999 character long string to hold the zones. Thus if the zone for zip 145 is 7, the 145th character in the string will be "7". You have to create your own 999 character long string, since the zones are dependent on your location. However, we are including the zone string that we use here so you can have an example.

Create Zone String

The first program listed below allows you to create your own 999 character zone string. It takes advantage of the fact that the zips are grouped into areas of a similar zone. You just specify the start and end zips, the program puts the zone into each position in the specified range. Here is how it works:

```
ZIP Start - ZIP End - ZONE
---         ---       --
1           9         0
10          43        5
44          44        6
45          etc...
```

```
init // program name is "upszones"
instructions
process
store(zipzone$,filename$)
//
PROC instructions
  PAGE
  PRINT "This program creates a data file"
  PRINT "for use in determining UPS zones."
  PRINT "Get out your UPS zone chart and"
  INPUT "hit <return> when ready": reply$
  PRINT
  PRINT "Type in the zip ranges (using the"
  PRINT "first 3 digits of the zip code)"
  PRINT "followed by the zone..."
ENDPROC instructions
```

more»

```
//
PROC init
  DIM reply$ OF 1 // for one character replies
  DIM zipzone$ OF 999, filename$ OF 20
  zipzone$(1:999):=""; filename$:="upszones.dat"
ENDPROC init
//
PROC process
  PRINT "ZIP Start - ZIP End - ZONE"
  PRINT "---         ---       --"
  start'zip:=1; end'zip:=1
  WHILE end'zip<999 DO
    LOOP
      TRAP
        PRINT AT 0,1: start'zip;
        INPUT AT 0,13,3: "": end'zip;
        IF end'zip>=start'zip THEN
          INPUT AT 0,23,1: "": zone'num
          EXIT
        ELSE
          PRINT " too low"
        ENDIF
      HANDLER
        PRINT " numbers only"
      ENDTRAP
    ENDLOOP
    FOR zips:=start'zip TO end'zip DO
      zipzone$(zips:zips):=STR$(zone'num)
    ENDFOR zips
    start'zip:=end'zip+1
    end'zip:=start'zip
  ENDWHILE
ENDPROC process
//
PROC store(text$,filename$)
  PRINT filename$;"...";
  IF file'exists(filename$) THEN
    INPUT "exists. Overwrite (y/n)?": reply$
    IF reply$="Y" OR reply$="y" THEN
      DELETE filename$
      PRINT "File";filename$;"deleted."
    ELSE
      PRINT "Data not stored."
      RETURN
    ENDIF
  ENDIF
  OPEN FILE 2,filename$,WRITE
  WRITE FILE 2: text$
  CLOSE FILE 2
  PRINT "Data stored in file."
ENDPROC store
//
FUNC file'exists(filename$) CLOSED
  TRAP
    OPEN FILE 5,filename$,READ
    CLOSE FILE 5
    RETURN TRUE
  HANDLER
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
```

Zone Chart Printer

The next program will print out a zone chart for you. It is printed in the same style as the UPS Zone Chart, so you can check if yours is accurate. A partial printout is shown below, followed by the program listing:

```
 10 -  43 ... 5
 44 ..... 6
 45 ..... 5
 46 -  47 ... 6
 48 -  89 ... 5
100 - 139 ... 5
etc...
```

```
init // program name is "upschart"
instructions
process
//
PROC init
  DIM zipzone$ OF 999
  printer$:="lp:"; screen$:="ds:"//C64,C128,CP/M
  //printer$:="lpt1:"; screen$:="con:"//Uni IBM
  //printer$:="lst:"; screen$:="con:"//Mytech IBM
  filename$:="upszones.dat"
  assign(zipzone$,filename$)
ENDPROC init
```

more»

```
//
PROC instructions
  PAGE
  PRINT "This program prints a zone chart"
  INPUT "Hit RETURN when printer is ready: ":
  reply$ // wrap line
ENDPROC instructions
//
PROC process
  SELECT OUTPUT printer$
  start'zone#:=0; start'zip:=1
  FOR zip:=1 TO 999 DO
    this'zone#:=VAL(zipzone$(zip:zip))
    IF this'zone#=start'zone# THEN
      NULL // keep going, still same
    ELSE
      IF start'zone#=0 THEN
        NULL //skip zone 0 codes
      ELSE
        IF zip>start'zip+1 THEN
          PRINT USING "### - ### ... #":
          start'zip,zip-1,start'zone# // wrap line
        ELSE
          PRINT USING "### ..... #":
          start'zip,start'zone# // wrap line
        ENDIF
      ENDIF
      start'zone#:=this'zone#
      start'zip:=zip
    ENDIF
  ENDFOR zip
  PRINT USING "### - ### ... #":
  start'zip,999,start'zone# // wrap line
  CLOSE
  SELECT OUTPUT screen$
ENDPROC process
//
PROC assign(REF text$,filename$)
  WHILE NOT file'exists(filename$) DO
    INPUT "Need"+filename$+"--> insert disk:":
    reply$ // wrap line
  ENDWHILE
  OPEN FILE 2,filename$,READ
  READ FILE 2: text$
  CLOSE FILE 2
ENDPROC assign
//
FUNC file'exists(filename$) CLOSED
  TRAP
    OPEN FILE 5,filename$,READ
    CLOSE FILE 5
    RETURN TRUE
  HANDLER
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
```

Address Label With Zone

Finally, here is a program that shows how you might use this zone calculator. The program prints a mail label for you, complete with your address, zone, and destination. An example is shown below, followed by the program listing:

```
FROM:
Your Name
Your Address
City, ST  12345

UPS ZONE 7

TO:
Calvin Turtle
1234 COMAL Light Way
COMAL City, CA  92600
```

```
init // program name is "demo/ups"
process
//
PROC init
  DIM reply$ OF 1 // for one character replies
  DIM name$ OF 30, company$ OF 30
  DIM address1$ OF 30, address2$ OF 30
  DIM city$ OF 20, state$ OF 2, zip$ OF 10
  DIM zipzone$ OF 999, filename$ OF 20
  filename$:="upszones.dat"
  assign(zipzone$,filename$)
  printer$="lp:";screen$="ds:"//C64,C128,CP/M
  //printer$:="lpt1:"; screen$:="con:" //Uni IBM
```

more»

```
  //printer$="lst:";screen$="con:"//Mytech IBM
ENDPROC init
//
PROC process
  LOOP
    get'person
    print'label
    continue
  ENDLOOP
ENDPROC process
//
PROC get'person
  PAGE
  INPUT "     name: ": name$
  INPUT "  company: ": company$
  INPUT " address1: ": address1$
  INPUT " address2: ": address2$
  INPUT "     city: ": city$
  INPUT "    state: ": state$
  INPUT "      zip: ": zip$(1:10)
ENDPROC get'person
//
PROC print'label
  SELECT OUTPUT printer$
  from'label
  zone'label
  to'label
  SELECT OUTPUT screen$
ENDPROC print'label
//
PROC from'label
  PRINT "FROM:"
  PRINT "Your Name"
  PRINT "Your Address"
  PRINT "City, ST  12345"
ENDPROC from'label
//
PROC zone'label
  PRINT
  TRAP
    temp:=VAL(zip$(1:3))
    PRINT "UPS ZONE";zipzone$(temp:temp)
  HANDLER
    PRINT "No Zone"
  ENDTRAP
  PRINT
ENDPROC zone'label
//
PROC to'label
  PRINT "TO:"
  print'it(name$)
  print'it(company$)
  print'it(address1$)
  print'it(address2$)
  PRINT city$,",";state$;zip$;
ENDPROC to'label
//
PROC print'it(text$) // skips null text
  IF text$>"" THEN PRINT text$
ENDPROC print'it
//
FUNC file'exists(filename$) CLOSED
  TRAP
    OPEN FILE 5,filename$,READ
    CLOSE FILE 5
    RETURN TRUE
  HANDLER
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
//
PROC assign(REF text$,filename$)
  WHILE NOT file'exists(filename$) DO
    INPUT "Need"+filename$+"--> insert disk:":
    reply$ // wrap line
  ENDWHILE
  OPEN FILE 2,filename$,READ
  READ FILE 2: text$
  CLOSE FILE 2
ENDPROC assign
//
PROC continue
  INPUT AT 24,1,1: "Hit <return> when ready
  (Q to Quit):": reply$ // wrap line
  IF reply$="Q" OR reply$="q" THEN halt
ENDPROC continue
//
PROC halt
  SELECT OUTPUT screen$
  END "All done."
ENDPROC halt ■
```

COMAL Flex

by Bill Nissley 

Comal-flex(joy) on *Today Disk #18* is a program for designing special fonts for use with the drawing program called *Flexidraw*. It was written primarily for the COMAL programmer to take advantage of these nice large size fonts in COMAL. Each character in this large font is actually made from four normal size characters. Along with this large font editor is a procedure to print to the screen in any large font available. The only limitation is that you end up with 63 characters. They normally are the letters a-z (upper or lowercase, but not both), the numbers 0-9, and the symbols !, ", #, $, %, &, ', (,), +, -, :, ;, =, *, /, <, >, ?, @, [,], period, comma, british pound sign, space, and raise arrow. You can create any special character you wish. The back arrow was omitted due special use of that key by COMAL and *Flexidraw*.

The first thing that happens is the program loads two fonts from disk. */times* is stored in an array and *^times* is the active font. After the main editing screen comes up you will see the 63 characters laid out on the left side of the screen and a large editing grid on the right side. This grid is where you can turn off and on pixels of the character you choose.

The keyboard is used to select the letter to be edited. Just press the key that is the letter you want to edit and it is put into the editing grid. Use the joystick in port 2 to move the cursor to the pixel you want to edit. The fire button flips the pixel.

The cursor keys shift the character one pixel in the direction of the key. All other options for creating fonts are accessed by the function keys. I have included a function key overlay for use with this program.

These commands are as follows:

F1) Reverses current letter on editing grid
F2) Rotates current letter 90 degrees to the left
F3) Mirrors current letter left to right
F4) Mirrors current letter up and down
F5) Stores the current letter in memory. (It then can be moved to any other character in the current font or in the other font that is stored in memory.)
F6) Used to recall the previously stored letter after the destination has been selected
F7) Switches the displayed font with the one stored in memory
F8) Is to access the disk drive

When using F5 and F6 you first select the character you want to copy by pressing that letter on the keyboard. Then you press F5, you will see the border flash signaling it has stored this character. Next you choose the destination for recalling the letter by using the keyboard or using F7 to access the other font and then selecting from the keyboard the letter desired. After the destination is selected you then press F6. If multiple copies are wanted then select another destination and press F6 again.

The F8 key has several commands in it. At this point, menu selections are made from the keyboard. They are:

1) Save to Flexidraw format
2) Load from Flexidraw format
3) Save to COMAL format
4) Load from COMAL format
D) Disk directory

The «*stop*» key is used to exit the program. It clears the screen and places you back in the normal upper case/ lower case font. The large *flexi* font may be obtained with the «*shift*»«*c=*» key. It replaces the upper case / graphics character set.

Big'print is a demo program on *Today Disk #18* which prints our address in large letters. It

more»

uses **PROC large, a CLOSED procedure** that lets you print to the text screen in a flexi size font. It is called to using this format:

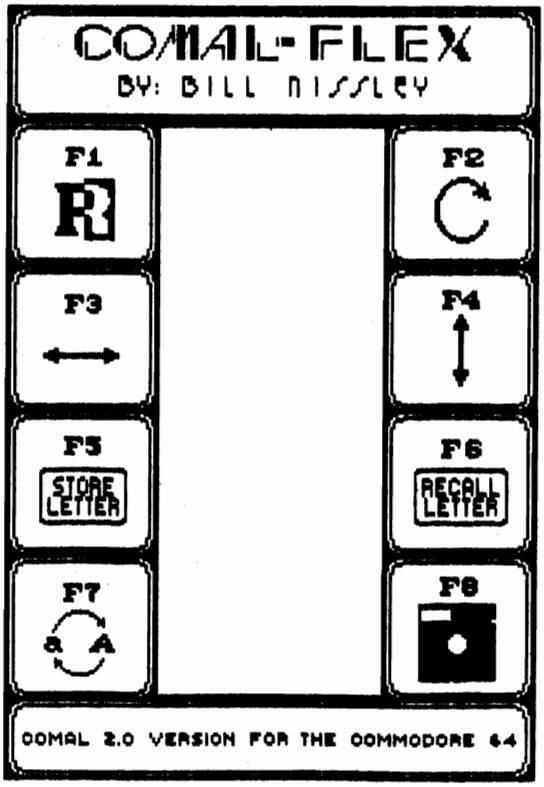
large (<row>,<column>,<text$>)

Row may be any row number between 1 and 13 the procedure automatically accounts for the added depth needed for these extra large characters.

Column is a number between 1 and 19. also by using a half number like 3.5 you can center an odd number of characters on the screen.

Text$ is a string variable or text inside quotes.

[Editor's note: Bill Nissley also sent us font editor which uses the lightpen instead of the joystick for input. Space limitations prevented us from placing it on Today Disk #18. If you would like the program, send us a blank disk or $2.00 and we will give you a copy.]



MT80 Dump

by Bernie Flinn

This hi-res screen dump package is for the Mannesmann Tally Spirit 80 printer. First, link the package into memory (and to your program if one resides in memory) from *Today Disk #18* by typing the following:

LINK "pkg.spiritdump"

If you wish the package to become part of your program, simply save the program in memory. To activate the package, type:

USE spiritdump

In order to dump the hi-res screen to the printer (in the immediate mode or in a program), type the command:

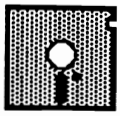
mt80dump(num)

Num specifies the type of screen dump:

value	type of dump
0	1/4 page
1	reverse 1/4 page
2	full page
3	reverse full page
4	double density 1/4 page
5	reverse double density 1/4 page
6	double density full page
7	reverse double density full page

The dump can be aborted by pressing the a key and holding it down until the current line being printed is completed. For a full page dump, the a key may have to be held down until two lines have been completely printed (due to the nature of the screen dump routine). Have faith though, the abort will work (it beats having to wait for 25 or more lines to be printed). ■

Print Shop / Print Master



by Terry R. Mills

After using *Print Shop* for more than a year, I recently purchased *PrintMaster Plus* from Unison World. After making some cards and such with it, I started thinking how neat it would be if I could use my *PrintMaster* graphics in *Print Shop* and vice versa. Not an original idea, to be sure. However, the programs advertised to do the job cost a lot more than the cost of a *Print Shop Graphics Library* disk or a *PrintMaster Art Gallery* disk. Surely, I thought, with a little study of the way the graphics are stored on disk, I could write my own conversion program in COMAL 0.14. And I was right!

Convert'ps/pm on *Today Disk #18* allows one to convert a *Print Shop* or *PrintMaster* graphic into the graphics format of the other program. The program itself is really quite elementary, thanks to COMAL's structured format. In fact, the trickiest part for me was figuring out that *PrintMaster* does not save the names of graphics in PETASCII and that *Print Shop* likes names all in one case (lower case, when viewing the directory in upper/lower case mode). This necessitated the name conversion routine in the **master'to'shop** procedure, but this too was no major problem because of COMAL's superior string handling.

To use the program, simply load and run it inside COMAL, insert the disk with the graphic to be converted,.and follow the simple prompts.

While I would guess that most people don't have both *Print Shop* and *PrintMaster* (or the new *PrintMaster Plus*), this program does make available to owners of either program the graphics libraries of both. Also, while I don't have the COMAL Iconmaker program, I would assume that this conversion program will allow COMALites to convert *PrintMaster* graphics and then edit them with it. In any event, I would like to share this program.

```
init
repeat
 menu
until c$="3"
//
proc init
 dim c$ of 1, k$ of 1, d$(52,11) of 1
 dim shop'name$ of 16, master'name$ of 16
 disk'get'init
endproc init
//
proc menu
 print chr$(14);chr$(147);
 print "Print Shop/PrintMaster Converter"
 print
 print "1 - Print Shop to PrintMaster"
 print
 print "2 - PrintMaster to Print Shop"
 print
 print "3 - Quit"
 print
 repeat
  c$:=key$
 until c$ in "123"
 if c$="1" then
  shop'to'master
 elif c$="2" then
  master'to'shop
 endif
endproc menu
//
proc shop'to'master
 input "Print Shop filename: ": shop'name$
 if shop'name$="" then end
 master'name$:=shop'name$+".gra"
 print
 print "Insert disk with Print Shop graphic"
 continue
 if not file'exists(shop'name$+",p") then
  message("Sorry, file not found.")
  abort:=true
 else
  abort:=false
  open file 1,shop'name$+",p",read
  print
  print "Print Shop graphic file is being read"
```

more»

```
  print "as counter advances to 52."
  print chr$(17)
  disk'get'skip(2,1,file'end)
  for i:=1 to 52 do
   print chr$(145);i
   for j:=1 to 11 do
    d$(i,j):=chr$(disk'get(1,file'end))
   endfor j
  endfor i
  close
 endif
 //
 if abort=false then
  print
  print "Insert disk you want converted"
  print "graphic to appear on"
  continue
  if file'exists(master'name$+",p") then
   message(shop'name$+" exists-not converted")
  else
   open file 2,master'name$+",p",write
   print
   print "PrintMaster graphic is being"
   print "written as counter advances to 52."
   print
   print file 2: chr$(80)+chr$(135),
   print file 2: chr$(88)+chr$(0),
   print file 2: chr$(52),chr$(0),
   print file 2: chr$(180),
   for i:=1 to 52 do
    print chr$(145);i
    print file 2: chr$(139),
    for j:=1 to 11 do
     print file 2: d$(i,j),
    endfor j
   endfor i
   close
  endif
 endif
endproc shop'to'master
//
proc master'to'shop
 print "When entering PrintMaster filename,"
 print "omit .gra extension."
 print
 input "PrintMaster filename: ": master'name$
 if master'name$="" then end
 shop'name$:=master'name$
 for i:=1 to len(master'name$) do
  a:=ord(master'name$(i))
  if a>192 and a<224 then
   a:=a-96 // pm - convert to true ascii
   b:=a-32 // ps - convert to lower case
   master'name$(i):=chr$(a)
   shop'name$(i):=chr$(b)
  endif
 endfor i
 master'name$:=master'name$+".gra"
 print
 print "Insert disk with PrintMaster graphic"
 continue
 if not file'exists(master'name$+",p") then
  message("Sorry, file not found.")
  abort:=true
 else
  abort:=false
  open file 1,master'name$+",p",read
  disk'get'skip(8,1,file'end)
  print
  print "PrintMaster graphic file is being read"
  print "as counter advances to 52."
  print chr$(17)
  for i:=1 to 52 do
   print chr$(145);i
   for j:=1 to 11 do
    d$(i,j):=chr$(disk'get(1,file'end))
   endfor j
   disk'get'skip(1,1,file'end)
  endfor i
  close
 endif
 //
 if abort=false then
  print
  print "Insert disk you want converted"
  print "graphic to appear on"
  continue
  if file'exists(shop'name$+",p") then
   message(master'name$+" exists-not
   converted") // wrap line
  else
   open file 2,shop'name$+",p",write
```

more»

```
    print
    print "Print Shop graphic is being written"
    print "as counter advances to 52."
    print chr$(17)
    print file 2: chr$(0),chr$(88),
    for i:=1 to 52 do
     print chr$(145);i
     for j:=1 to 11 do
      print file 2: d$(i,j),
     endfor j
    endfor i
    for i:=1 to 5 do
     print file 2: chr$(0),
    endfor i
    close
   endif
  endif
endproc master'to'shop
//
proc disk'get'init closed
 for location:=2024 to 2039 do
  read value
  poke location,value
 endfor location
 data 0,162,0,32,198,255,32,207
 data 255,141,232,7,32,204,255,96
endproc disk'get'init
//
proc disk'get'skip(count,file'num,ref file'end)
closed // wrap line
 for x:=1 to count do
  y:=disk'get(file'num,file'end)
 endfor x
endproc disk'get'skip
//
func disk'get(file'num,ref file'end) closed
 poke 2026,file'num
 sys 2025
 file'end:=peek(144)
 return peek(2024)
endfunc disk'get
//
func file'exists(file'name$) closed
 dim s$ of 2
 open file 78,file'name$,read
 s$:=status$
 close file 78
 return s$="00"
endfunc file'exists
//
proc continue
 print "and press RETURN."
 repeat
  k$:=key$
 until k$=chr$(13)
endproc continue
//
proc message(text$)
 print text$
 for delay:=1 to 3000 do null
endproc message ■
```

Display Print Shop Pictures



by Craig Hardy

Pkg.ps'convert loads a *Print Shop* file into COMAL 2.0 and displays the bitmap at twice its size. Use only the 3 block long *Print Shop* files on side A of the *Print Shop* disks.

By using the **savescreen(filename$)** command you can save these bitmaps for use in your own personal programs or you may want to convert them to another format (such as *Doodle*) for editing. The package adds two commands: **get'file(filename$)** and **put'screen.**

I located the package at $B800 to avoid possible conflicts with other packages. The program which contains this package is called *printshop'loader*. Both are on *Today Disk #18.*

[Editor's note: the 2.0 side of Today Disk #18 contains two Print Shop Icons: icon.garf'head and icon.donald'duck. These may be loaded into COMAL 2.0 with Craig Hardy's printshop'loader program or converted to Print Master format by Terry Mills' COMAL 0.14 program.] ■

Multiple Programs



by Richard Bain

David Vosh recently wrote a letter to us asking about the possibility of having two or more COMAL programs in memory at once. He cited several magazine articles showing how BASIC handled the task. In particular, he wanted to have at least two COMAL 0.14 programs running together with their own set of independent variables. He thought it would be helpful to use the jiffy clock to allocate time slices to each program, perhaps one would get two thirds of the computers attention while the other got one third. Even if the COMAL programs only had 2K bytes free, the multitasking system could be used effectively for educational purposes.

The first thing we had to do was simplify his request. We would work in COMAL 2.0 instead of 0.14. Instead of using the jiffy clock to transfer control, we would use a package procedure. Now for a little background on how we solved the problem.

We had just received CP/M COMAL and learned about its **USER** area and the **EXTERNAL** section. The **USER** area is created with the **NEW** command:

NEW 500

This command reserves 500 bytes for use by COMAL programs. Data or machine language routines can be placed in the **USER** section without conflict from CP/M or the COMAL system. The built in function, **USER**, returns the address of the first byte of the **USER** section.

The **EXTERNAL** command in CP/M COMAL allows several programs in memory at once. It is often desirable to write a short program to test something while writing a long program. It can be inconvenient to have to **SAVE** your work, perform the test, and **LOAD** the original program back in. It can also be a pain to write the test routine at the start of the program and use the **END** command to prevent the main program from executing after your test is complete. CP/M COMAL solved this problem with the **EXTERNAL** command. When the main program is in memory, just enter **EXTERNAL** from the command mode. This initializes a section of memory for your test program. When the test is complete, type **MAIN** to go back to the main program. The only weakness in this system is that once you type **MAIN**, the **EXTERNAL** program is lost forever (unless you saved it to disk).

It was in the spirit of these commands that we modeled our memory management package for the COMAL 2.0 cartridge. The heart of the package taps into the **NEW** command. It resets the system variables which point to the start and end of the COMAL program area (the defaults are **$0800** and **$7fff**). To make this useful, commands are included to determine where the program will be located the next time the **NEW** command is executed. There are other commands to ease the process of switching between programs, either during command mode or from running programs.

This system is not very user friendly, nor does it have the error checking necessary to prevent a complete system crash. Therefore, I have included a batch file that gives an example of how to correctly use the package. However, I will first give a list of the commands included in the package.

LINK "pkg.memory"
This command links in the memory package from *Today Disk #18*. The package is a ROMMED package located from $8009 to $8453. It can (and must) survive in memory after the **NEW** command is issued.

USE memory
This command enables the package procedures. It must be placed at the start

more»

of all programs which use any of the following commands.

program'base(first'address)
This procedure sets the address where the next program will start after NEW is issued. The number must be at least $0800. It does not effect the current program.

program'top(last'address)
This procedure sets the highest address that the next program will use after NEW is issued. The number must not be greater than $7fff. It does not effect the current program.

NEW
This COMAL command has been altered to locate the next program between the addresses defined above. These addresses will remain for typed in programs or programs loaded from disk. The addresses even survive the «*run/stop*»«*restore*» combination. The only way to reset them is to issue the **USE memory** command again and then reset the addresses as described above. This is a safe way to reserve memory for interrupt driven machine code, sprite images intended to be used with the text screen, and the like. Warning, there is no error checking that the addresses are valid, so it is possible to lock up the computer with this command.

bye
This command is like the NEW command in that it sets up a fresh program segment as described above. The difference is that it also saves system information to let it return to the current program segment. The intent is to first create one program segment with the altered NEW command. Then set up a second memory area which does not overlap the first one. Use the **bye** command to exit from the first program segment to the second one, leaving the possibility to return back to the first.

switch
This command switches control from the current program to the alternate program which was last exited with the **bye** command or a previous **switch** command. System information is saved to make it possible to return to the current program. If **switch** is called from within a running program, program execution will continue on the following line when control is switched back. If **switch** is called from command mode and **CON** was possible before the call, **CON** should still be possible when control is passed back to the program. Warning, no error checking is provided to ensure that you **switch** to a valid program segment.

get'mem(info$)
This command is used fetch system information which was saved with the last **bye** or **switch** command into the string info$. The string must have been dimensioned to at least 410 characters. This command is intended to be used with the **set'mem(info$)** command by a special control program. It makes it possible for one program to pass control to any of several other programs in memory. The other program can **switch** control back to special control program which may then select a third program to run. This example is demonstrated from the batch file at the end of the article.

set'mem(info$)
This command is the complement of **get'mem**. You must first initialize info$ with the **get'mem(info$)** command. Then this command can be used to restore that system information.

more»

version'memory$
This function returns the string containing the package version information.

Warning about the memory package: This package has not been fully tested and debugged. We do not know exactly which COMAL system variables are required to fully represent a program. We tried to save any pointers we thought might be important, but we may have missed a few. In particular, the package does not save the file information. If you plan to do any file access with programs using this package, only use temporary disks which contain no important files.

The following text is a batch file on *Today Disk #18* along with the memory package and its source code. It creates three program segments which run together in memory. One program is the master control program which calls the other two. The other two require you to press a key before they **PRINT** a message and return to the control program. To execute the batch file, type:

SELECT INPUT "bat.memory"

The following lines will be entered from disk:

```
link"pkg.memory"
use memory
program'base($7000)
program'top ($7fff)
new
10 use memory
20 dim prog$(2) of 410
30 program'base($0800)
40 program'top($3fff)
50 stop
60 get'mem(prog$(1))
70 program'base($4000)
80 program'top($6fff)
90 stop
100 get'mem(prog$(2))
110 choose
120 //
130 proc choose
140 loop
150 input "program 1 or 2? ":x
160 set'mem(prog$(x))
170 switch
180 get'mem(prog$(x))
190 endloop
200 endproc choose
run
bye
10 use memory
20 loop
30 print "You are in program 1"
40 print "Press any key to continue"
50 while key$<=chr$(0) do null
60 switch
70 endloop
run
«space»
con
bye
10 use memory
20 loop
30 print "You are in program 2"
40 print "Press any key to continue"
50 while key$<=chr$(0) do null
60 switch
70 endloop
run
«space»
con
```

This program is purely tutorial in nature. Try running it and choosing program 1 or 2. Then hit a key to start over. After a while try hitting the «*stop*» key and listing one of the programs. Then type **CON** to continue. When you really feel secure with what is going on, enter program 1, hit «*stop*», and type **NEW**. Then **LOAD** and **RUN** another program. Next, type **USE memory** and **switch**. You will be back in the control program and be prompted to enter program 1 or 2. Let us know of any useful examples for this package. ■

Third Dimension

Paul van Leeuwen, Dutch COMAL Users Group

One of the nice aspects of using computers is drawing pictures. Because the laws of perspective can be formulated in mathematical terms, you can program the computer to draw 3-dimensional pictures.

I previously thought that extended and complex programs were needed to do this. However, while reading a German magazine, *C64'er*, I discovered a small program with impressive and, above all, nice pictures for the C64.

The program was written in BASIC, but translating it into COMAL was easy. Now it runs twice as fast. *3rd'dimension* on *Today Disk #18* draws a picture based on a function, such as:

```
FUNC f(displacement)
  RETURN 60*SIN(displacement/30)+10
ENDFUNC f
```

For plotting we use x,y,z-coordinates with a horizontal x,y-plane. The vertical z-coordinate is the value of a function **f(displacement)**. The **displacement** is the distance from (x,y)=(0,0) or the z-axis.

```
init
graf
plottext(24,8,"Press <space> to continue") // 2.0
//plottext 24,8,"press <space> to continue" //.14
WHILE KEY$<>" " DO NULL
//
FUNC f(displacement)
  RETURN 60*SIN(displacement/30)+10
ENDFUNC f
//
PROC init
  USE graphics // 2.0 only
  graphicscreen(0) // 2.0 only
  // setgraphic 0 // 0.14 only
  background(3)
  border(3)
  pencolor(1)
  x'center:=160
  y'offset:=60
ENDPROC init
//
PROC graf
  FOR x:=0 TO -100 STEP -1 DO
    top:=-y'offset
    radius:=5*INT(SQR(10000-x*x)/5)
    FOR y:=radius TO -radius STEP -5 DO
      height:=25+f(SQR(x*x+y*y))-.6*y
      IF height>top THEN
        top:=height
        plot(x'center+x,y'offset+top) //2.0
        plot(x'center-x,y'offset+top) //2.0
        // plot x'center+x,y'offset+top //0.14
        // plot x'center-x,y'offset+top //0.14
      ENDIF
    ENDFOR y
  ENDFOR x
ENDPROC graf
```



We draw a circle symmetric function around the z-axis, based on the function **f**. Because we want to see this drawing from above, all points lying before the x,z-plane are shifted down, points behind the x,z-plane are shifted up. This is done with the term **-.6*y** in:

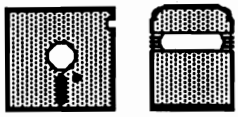
```
height:=25+f(SQR(x*x+y*y))-.6*y
```

Because all points are plotted upwards in the y,z-plane (the screen), it is simple to take care of hidden points. The old value of **height** is stored in the help variable **top**. After comparing these two values we can **PLOT** points on *top* of other points, but **not** **PLOT** points *behind* or *underneath* other points.

Some other interesting functions **f** are:

```
15*ABS(LOG/70+0.5)-LOG(x*20+0.5)
INT(x)+110
90*EXP(-x*x/1500)
85/(x/25+1)
30*COS(PI*x/15) ■
```

Fractal Geometry



by Ted Groszkiewicz

One of the first projects I undertook after receiving my COMAL 2.0 cartridge was the translation of my Logo recursive pattern collection. It was most gratifying to watch the increased speed of execution.

Recently my brother introduced me to the intricacies of Benoit Mandelbrot's Fractal Geometry. Mandelbrot actually coined the term fractal geometry - the mathematical study of forms having a fractional dimension.

This article focuses on the investigation of the **Mandelbrot Set**, a particular set of complex numbers. A complex number is one that possesses both a real and an imaginary component. A complex number takes the form a+bi where i, the imaginary number, equals (-1)^.5. The *size* of an imaginary number can be stated as (a^2 + b^2)^.5. The numbers in the Mandelbrot Set are distinguished by their mathematical behavior. If a mathematical operation is carried out repeatedly on most numbers, the size of the result eventually grows to infinity. For numbers in the Mandelbrot Set the size of the result never exceeds 2.

To understand the math involved let's review addition and multiplication of complex numbers:

Addition: (a+bi)+(c+di) = (a+c)+(b+d)i

Multiplication: (a+bi)*(c+di) = (ac-bd)+(ad+bc)i

Suppose we use two complex numbers: a variable number z and a fixed number c. Initially we'll set z=0 and then iterate the operation z=z^2 + c:

Iteration	Result
1	z=c
2	z=c^2 + c
3	z=(c^2+c)^2 + c
4	z=((c^2+c)^2+c)^2 + c

If the number c is in the Mandelbrot Set z will never exceed 2 no matter how many times the expression z=z^2+c is iterated.

The complex number c may be located on a set of coordinates whose horizontal (x) axis is the real argument and whose vertical (y) axis is the imaginary argument. The number of iterations of the mathematical operation necessary for the size of z to reach 2 is assigned a color value and the point is plotted. The resulting image is rather like a contour map. The two illustrations accompanying this article are screen dumps of actual program runs.

The programs *mandelbrot* and *mandelbrot'small* prompt the user to pick a real and an imaginary argument (center) for c and a range over which to vary c's real and imaginary arguments (there is an autoscale option for the imaginary range). Prompts for saving the screen image and for inputting the number of iterations also appear. Then the program goes to work.

Mandelbrot uses the entire multicolor screen. While its images are large and detailed, a 50 iteration run in the vicinity of the Mandelbrot Set can take upwards of 12 hours to complete. The 160 x 200 screen map could require over 6 million separate calculations and a like number of logical evaluations. *Mandelbrot'small* only uses a quarter of the screen (and time). It is a good tool for locating interesting areas for exploration. There is an apparent infinite diversity in the region surrounding the actual Mandelbrot Set. Suggestions regarding coding the calculation module in machine code for faster processing are most welcome. A box cursor that could set the boundaries for a subsequent run would also be nice.

I am interested in corresponding with any readers whose interest is piqued by this very intriguing phenomenon. You can write me at: Ted Groszkiewicz, 72 Bryson Heights, Waynesville, NC 28786.

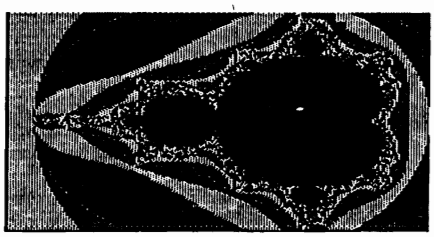
more»

```
// Commodore 2.0 version
setup
graphicscreen(1)
background(0); border(0)
calculate
IF name$<>"" THEN savescreen(name$)
//
PROC setup
  USE graphics
  PAGE
  DIM ans$ OF 1
  INPUT "Real Center: ": real'center
  INPUT "Real Range: ": real'range
  INPUT "Imaginary Center: ": im'center
  INPUT "Autoscale Imaginary range y/n?": ans$
  IF ans$="n" OR ans$="N" THEN
    INPUT "Imaginary Range: ": im'range
  ELSE
    im'range:=real'range*.77
  ENDIF
  INPUT "Save the screen at end (y/n)? ": ans$
  IF ans$="y" OR ans$="Y" THEN
    INPUT "Filename? ": name$
  ENDIF
  INPUT "How Many Iterations? ": it
ENDPROC setup
//
PROC calculate
  xs:=real'center-real'range/2
  ys:=im'center-im'range/2
  gapx:=real'range/318
  gapy:=im'range/199
  FOR xp#:=2 TO 320 STEP 2 DO
    x:=xp#*gapx+xs
    FOR yp#:=1 TO 200 DO
      y:=yp#*gapy+ys
      count:=0; az:=0; bz:=0; ac:=x; bc:=y
      sizez:=0
      mandelbrot
    ENDFOR yp#
  ENDFOR xp#
ENDPROC calculate
//
PROC dot
  b:=(count MOD 3)
  pencolor(b+6)
  IF count>=it THEN pencolor(0)
  plot(xp#,yp#)
ENDPROC dot
//
PROC mandelbrot
  WHILE count<it AND sizez<2 DO
    count:+1
    IF count>=it THEN dot
    ad:=az*az-bz*bz; bd:=2*az*bz
    az:=ad+ac; bz:=bd+bc
    sizez:=SQR(az*az+bz*bz)
    IF sizez>=2 THEN dot
  ENDWHILE
ENDPROC mandelbrot

// Commodore 0.14 version
setup
border 0
background (0)
setgraphic 1
hideturtle
calculate
//
proc setup
 print chr$(147),chr$(14),
 dim ans$ of 1
 input "Real Center: ": real'center
 input "Real Range: ": real'range
 input "Imaginary Center: ": im'center
 input "Autoscale Imaginary Range (y/n)?": ans$
 if ans$="n" or ans$="N" then
  input "Imaginary Range: ": im'range
 else
  im'range:=real'range*.77
 endif
 input "How Many Iterations? ": it
endproc setup
//
proc calculate
 xs:=real'center-real'range/2
 ys:=im'center-im'range/2
 gapx:=real'range/318
 gapy:=im'range/199
 for xp#:=2 to 320 step 2 do
  x:=xp#*gapx+xs
  for yp#:=1 to 200 do
```

more»

```
    y:=yp#*gapy+ys
    count:=0; az:=0; bz:=0; ac:=x; bc:=y
    sizez:=0
    mandelbrot
   endfor yp#
  endfor xp#
endproc calculate
//
proc dot
 b:=(count mod 3)
 pencolor (b+6)
 if count>=it then pencolor (0)
 plot xp#,yp#
endproc dot
//
proc mandelbrot
 while count<it and sizez<2 do
  count:+1
  if count>=it then dot
  ad:=az*az-bz*bz; bd:=2*az*bz
  az:=ad+ac; bz:=bd+bc
  sizez:=sqr(az*az+bz*bz)
  if sizez>=2 then dot
 endwhile
endproc mandelbrot
```

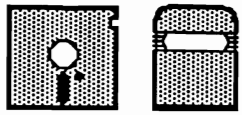


real center: -.75, real range: 2.5
imaginary center: 0, autoscale: y



real center: -.75, real range: 2.5
imaginary center: 0, autoscale: y ■

Blackjack



by Jeffrey Sherwood

In preparation for a trip to Las Vegas, I read the best selling book *Beat The Dealer* by Edward O. Thorp. I found the examination of the game of Blackjack to be very interesting. The author describes how he used probability theory and a "high speed IBM 701 computer" to work out a winning strategy. The book gives several methods of play that involve counting certain cards and examining ratios of some cards to others. But first, the reader must learn a basic strategy based only upon the player's card totals and the dealer's up card.

Blackjack'advice on *Today Disk #18* pits you against the dealer in one of the world's most widely played gambling games. It is said that the typical casino player has a 2% to 15% disadvantage against the house. However, by using the basic strategy he improves up to a 1% advantage. I decided a good way to learn and practice the strategy would be to write a smart blackjack simulation. The resulting program may be used to learn the game as well as to study the more advanced strategies. The first inquiry the program makes is if you would like the help of the blackjack advisor. Reply Y for recommendations based on the basic strategy for standing, hitting, doubling down, or splitting pairs. The option of insurance is left to the player's discretion. If you play unassisted, you will still see what choice the advisor would have made.

The player makes a bet before each hand. Only bets of one to nine are allowed. While in help mode, press the space bar to make the recommended play. The number of card decks used is determined by the variable **dx** at the beginning of the program. Also, the program will play automatically if you set the variable **aut** to true. After seeing results of many hands, you may judge the effectiveness of using the basic strategy. ■

Knights Tour



by Bill Inhelder

Several years ago, before the advent of personal microcomputers, a friend suggested an interesting programming project: simulate a knight's tour. A knight's tour begins by starting at any position on a chess board and advancing the knight using legal L-shaped moves landing on each square once and only once. The tour is complete when all 64 squares are covered. If the tour can return to the starting position on the 65th move it is called a re-entrant tour. The object is to avoid moving to a square for which there is no exit.

An article by Donald P. Irwin in the May 1985 issue of *Creative Computing* presented a two-stage look-ahead method. It selects the move which has the **least** potential for future moves. In this way there is less chance of leaving a square which cannot be accessed. The rule uses a random move for the case where more than one minimum exit level exists. This provides for many different tours from the same starting point.

The program displays each move as it is made. The heart of the program involves an array of legal moves and the ability to determine the exit values for each of the potential moves from a given board position.

The program provides the following options:

1. user selected starting position
2. printer output of each tour

The program will produce many different tours using the same starting position. See if you can find a re-entrant tour. The screen dump below shows a re-entrant tour.

The following program works as listed for all COMAL 2.0 systems. However, the device names in the **print'out** procedure must be changed in order to print a completed tour from an IBM system. COMAL 0.14 users should see page 7 for special instructions to enter the program. *Knight's* (both the COMAL 2.0 version and the slightly modified 0.14 version) is on *Today Disk #18.*

2	39	16	23	32	37	14	25
17	22	1	38	15	24	33	36
40	3	62	31	44	35	26	13
21	18	41	64	61	30	43	34
4	63	20	57	42	45	12	27
19	54	51	60	29	58	9	46
50	5	56	53	48	7	28	11
55	52	49	6	59	10	47	8

```
PAGE
PRINT AT 12,15: "Knight's Tour"
PRINT AT 14,25: "By"
PRINT AT 15,25: "Bill Inhelder"
DIM legal'pos#(8)
DIM legal'move$(64) OF 64, exit'value#(64),
chess'brd'pos#(8,8)
DIM reply$ OF 1
INPUT AT 20,1,1: "hit <return> to start": reply$
RANDOMIZE // not needed by Mytech
REPEAT
  chess'board
  number'board
  INPUT AT 23,1: "Enter knight's starting
  position (0-64):": n // wrap line
  WHILE n>0 DO
    PAGE
    PRINT AT 12,12: "Setting Up Arrays"
    read'data
    init'arrays
    chess'board
    knight'pos'count:=1
```

more»

```
    display'move
    coord'translate
    next'move'algorithm
  ENDWHILE
  IF n<>0 THEN
    INPUT AT 23,1: "Printout? (Y / N):": reply$
    IF reply$="Y" OR reply$="y" THEN print'out
  ENDIF
UNTIL n=0
//
PROC chess'board
  PAGE
  PRINT AT 3,8: "+--+--+--+--+--+--+--+--+"
  FOR i#:=1 TO 8 DO
    PRINT AT 2*i#+2,8: "!",
    FOR j#:=1 TO 8 DO PRINT "  !",
    PRINT AT 2*i#+3,8: "+",
    FOR j#:=1 TO 8 DO PRINT "--+",
  ENDFOR i#
ENDPROC chess'board
//
PROC number'board
  FOR i#:=1 TO 64 DO
    IF i# MOD 8=0 THEN
      row:=(i# DIV 8+1)*2
      col:=30
    ELSE
      row:=(i# DIV 8+2)*2
      col:=i# MOD 8*3+6
    ENDIF
    PRINT AT row,col: USING "##": i#
  ENDFOR i#
ENDPROC number'board
//
PROC init'arrays
  FOR i#:=1 TO 8 DO
    FOR j#:=1 TO 8 DO
      chess'brd'pos#(i#,j#):=0
    ENDFOR j#
  ENDFOR i#
  FOR i#:=1 TO 8 DO legal'pos#(i#):=0
  FOR i#:=1 TO 64 DO exit'value#(i#):=0
  FOR j#:=1 TO 64 DO
    FOR i#:=1 TO 64 DO
      exit'value#(j#):=exit'value#(j#)+ORD(
      legal'move$(i#)(j#:j#)) // wrap line
    ENDFOR i#
  ENDFOR j#
ENDPROC init'arrays
//
PROC display'move
  IF n MOD 8=0 THEN
    row:=(n DIV 8+1)*2
    col:=30
  ELSE
    row:=(n DIV 8+2)*2
    col:=n MOD 8*3+6
  ENDIF
  PRINT AT row,col: USING "##":
  knight'pos'count // wrap line
ENDPROC display'move
//
PROC coord'translate
  r:=INT(n/8.01)+1
  c:=n-(r-1)*8
  chess'brd'pos#(r,c):=knight'pos'count
ENDPROC coord'translate
//
PROC next'move'algorithm
  REPEAT
    FOR i#:=1 TO 64 DO
      IF ORD(legal'move$(n)(i#:i#))=1 THEN
        exit'value#(i#):=exit'value#(i#)-1
        legal'move$(n)(i#:i#):=CHR$(0)
      ENDIF
    ENDFOR i#
    min'exit'val#:=10
    p:=1
    num'legal'moves:=0
    FOR i#:=1 TO 64 DO
      IF ORD(legal'move$(i#)(n:n))=1 THEN
        num'legal'moves:+1
        IF exit'value#(i#)<min'exit'val# THEN
          min'exit'val#:=exit'value#(i#)
          legal'pos#(p):=i#
        ELIF exit'value#(i#)=min'exit'val# THEN
          p:=p+1
          legal'pos#(p):=i#
        ENDIF
      ENDIF
    ENDFOR i#
    IF num'legal'moves<>0 THEN
```

more»

```
      IF p<>1 THEN
        p:=RND(1,p) // p:=RND#(1,p) for Mytech
      ENDIF
      r:=INT(legal'pos#(p)/8.01)+1
      c:=legal'pos#(p)-(r-1)*8
      knight'pos'count:+1
      IF knight'pos'count<=64 THEN
        chess'brd'pos#(r,c):=knight'pos'count
        n:=legal'pos#(p)
        FOR u:=1 TO 8 DO
          legal'pos#(u):=0
        ENDFOR u
        display'move
      ENDIF
    ENDIF
    IF num'legal'moves=0 AND knight'pos'count
    <64 THEN // wrap line
      PRINT AT 21,1: "Incomplete Tour!"
      FOR k:=1 TO 2000 DO NULL
      n:=-1
    ENDIF
    IF num'legal'moves=0 AND knight'pos'count
    =64 THEN // wrap line
      PRINT AT 21,1: "Tour Complete!"
      n:=-1
    ENDIF
    a:=num'legal'moves
    b:=knight'pos'count
  UNTIL a=0 AND b<64 OR a=0 AND b=64
ENDPROC next'move'algorithm
//
PROC print'out
  printer$:="lp:"; screen$:="ds:" // IBM is dif
  PRINT AT 21,1: "***TURN ON PRINTER***"
  PRINT AT 23,1: SPC$(39)
  INPUT AT 23,1: "hit <return> to start": reply$
  SELECT OUTPUT printer$
  FOR k:=1 TO 8 DO
    FOR l:=1 TO 8 DO
      PRINT USING "####": chess'brd'pos#(k,l);
    ENDFOR l
    PRINT
    PRINT
    PRINT
  ENDFOR k
  PRINT
  PRINT
  SELECT OUTPUT screen$
ENDPROC print'out
//
PROC read'data
  legal'move$(1):=SPC$(64)
  FOR i#:=1 TO 64 DO legal'move$(1)(
  i#:i#):=CHR$(0) // wrap line
  FOR i#:=2 TO 64 DO legal'move$(i#
  ):=legal'move$(1) // wrap line
  RESTORE
  FOR i#:=1 TO 336 DO
    READ row,col
    legal'move$(row)(col:col):=CHR$(1)
  ENDFOR i#
  DATA 11,1,18,1,12,2,17,2,19,2,9,3
  DATA 13,3,18,3,20,3,10,4,14,4,19,4
  DATA 21,4,11,5,15,5,20,5,22,5,12
  DATA 6,16,6,21,6,23,6,13,7,22,7,24
  DATA 7,14,8,23,8,3,9,19,9,26,9,4
  DATA 10,20,10,25,10,27,10,1,11,5
  DATA 11,17,11,21,11,26,11,28,11,2
  DATA 12,6,12,18,12,22,12,27,12,29
  DATA 12,3,13,7,13,19,13,23,13,28
  DATA 13,30,13,4,14,8,14,20,14,24
  DATA 14,29,14,31,14,5,15,21,15,30
  DATA 15,32,15,6,16,22,16,31,16,2
  DATA 17,11,17,27,17,34,17,1,18,3
  DATA 18,12,18,28,18,33,18,35,18,2
  DATA 19,4,19,9,19,13,19,25,19,29
  DATA 19,34,19,36,19,3,20,5,20,10
  DATA 20,14,20,26,20,30,20,35,20
  DATA 37,20,4,21,6,21,11,21,15,21
  DATA 27,21,31,21,36,21,38,21,5,22
  DATA 7,22,12,22,16,22,28,22,32,22
  DATA 37,22,39,22,6,23,8,23,13,23
  DATA 29,23,38,23,40,23,7,24,14,24
  DATA 30,24,39,24,10,25,19,25,35,25
  DATA 42,25,9,26,11,26,20,26,36,26
  DATA 41,26,43,26,10,27,12,27,17,27
  DATA 21,27,33,27,37,27,42,27,44,27
  DATA 11,28,13,28,18,28,22,28,34,28
  DATA 38,28,43,28,45,28,12,29,14,29
  DATA 19,29,23,29,35,29,39,29,44,29
  DATA 46,29,13,30,15,30,20,30,24,30
```

more»

Knight's Tour - continued

```
  DATA 36,30,40,30,45,30,47,30,14
  DATA 31,16,31,21,31,37,31,46,31
  DATA 48,31,15,32,22,32,38,32,47,32
  DATA 18,33,27,33,43,33,50,33,17
  DATA 34,19,34,28,34,44,34,49,34
  DATA 51,34,18,35,20,35,25,35,29,35
  DATA 41,35,45,35,50,35,52,35,19
  DATA 36,21,36,26,36,30,36,42,36
  DATA 46,36,51,36,53,36,20,37,22
  DATA 37,27,37,31,37,43,37,47,37
  DATA 52,37,54,37,21,38,23,38,28
  DATA 38,32,38,44,38,48,38,53,38
  DATA 55,38,22,39,24,39,29,39,45
  DATA 39,54,39,56,39,23,40,30,40
  DATA 46,40,55,40,26,41,35,41,51
  DATA 41,58,41,25,42,27,42,36,42
  DATA 52,42,57,42,59,42,26,43,28
  DATA 43,33,43,37,43,49,43,53,43
  DATA 58,43,60,43,27,44,29,44,34
  DATA 44,38,44,50,44,54,44,59,44
  DATA 61,44,28,45,30,45,35,45,39
  DATA 45,51,45,55,45,60,45,62,45
  DATA 29,46,31,46,36,46,40,46,52
  DATA 46,56,46,61,46,63,46,30,47
  DATA 32,47,37,47,53,47,62,47
  DATA 64,47,31,48,38,48,54,48,63
  DATA 48,34,49,43,49,59,49,33,50
  DATA 35,50,44,50,60,50,34,51,36
  DATA 51,41,51,45,51,57,51,61,51
  DATA 35,52,37,52,42,52,46,52
  DATA 58,52,62,52,36,53,38,53,43
  DATA 53,47,53,59,53,63,53,37,54
  DATA 39,54,44,54,48,54,60,54,64
  DATA 54,38,55,40,55,45,55,61,55
  DATA 39,56,46,56,62,56,42,57
  DATA 51,57,41,58,43,58,52,58,42
  DATA 59,44,59,49,59,53,59,43,60
  DATA 45,60,50,60,54,60,44,61,46
  DATA 61,51,61,55,61,45,62,47,62
  DATA 52,62,56,62,46,63,48,63
  DATA 53,63,47,64,54,64
ENDPROC read'data ■
```

Edit Source



by Dick Klingens

Working with Commodore's assembler editor only for adding some notes to the assembler statements, I could not suppress the feeling that is was possible to do same with a small COMAL program. This would avoid the long process listed below:

- first leave COMAL, back to BASIC
- find the assembler
- load the editor
- change disks
- load the source
- make changes
- scratch original file
- save changed file
- leave BASIC and return to COMAL

Edit'src, the result of an evening's work, is present on *Today Disk #18*. Some features of the program:

- a full screen editor
- page up and down
- fast scroll up and down
- move to last and first page
- copy lines
- delete to end of line
- tab stops
- easy load and save

It is a source editor, and is usable as a text processor, but there are some limitations:

- maximum line length of 39
- numbers of lines limited to 450
- no direct deleting of lines
- inserting lines not implemented
- no upper case
- double quote not possible

More information about how to use the program is included on its help screen. Press «*f7*» to see the help screen. ■

Reveal Procs 

by Dick Klingens, Dutch COMAL Users Group

COMAL was developed for educational purposes: teaching programming and an introduction to structured problem solving. Writing a program with many small modules can be exciting. And now, the **procs** procedure listed below will give you a summary of all your procedures and functions. To use **procs**, first type in the procedure as listed (it includes the nested function **proc'name$**). Next **LIST** it to disk:

```
LIST "proc.procs"
```

Now, this procedure can be merged onto any of your programs. For example, if you have a program called **towers** on disk, do this:

```
LOAD "towers"
MERGE "proc.procs"
```

Make sure the right disk is in the drive for each command. Now, to see a summary of all the procedures (and functions) in the program:

```
SCAN
PROCS
```

This procedure is compatible with *hidden lines* as discussed in the **Rod the Roadman** article in *COMAL Today #9*. Hidden procedures are not shown. This way, a teacher can provide a set of procedures for the student a hidden lines, and those procedures will not be listed by **procs**. *Today Disk #18* includes the **procs** procedure as well as a demo program.

```
PROC procs CLOSED
  start:=$0800 // of COMAL program
  REPEAT
    hilin#:=PEEK(start)
    lolin#:=PEEK(start+1)
    linno:=hilin#*256+lolin#
    length#:=PEEK(start+2)
    token#:=PEEK(start+3)
    CASE token# OF
    WHEN 112 // found procedure
      looff#:=PEEK(start+4)
      hioff#:=PEEK(start+5)
      namoff:=hioff#*256+looff#
      fn$:=proc'name$(namoff)
      IF linno>10000 THEN // not hidden
        PRINT " PROC ",fn$
      ENDIF
    WHEN 227,228,229 // found function
      looff#:=PEEK(start+4)
      hioff#:=PEEK(start+5)
      namoff:=hioff#*256+looff#
      fn$:=proc'name$(namoff)
      IF linno>10000 THEN // not hidden
        PRINT " FUNC ",fn$,
        // followed by type character
        CASE token# OF
        WHEN 227 //     real func
          PRINT ""
        WHEN 228 //  integer func
          PRINT "#"
        WHEN 2 //     string func
          PRINT "$"
        ENDCASE
      ENDIF
    OTHERWISE // found something else
    ENDCASE
    // next item in name table
    start:+length#
  UNTIL length#=0    end of table
  //
  FUNC proc'name$(ad) CLOSED
    // ad is offset in name table
    namtab:=$18 //pointer name table
    lonam#:=PEEK(namtab)
    hinam#:=PEEK(namtab+1)
    namstart:=hinam#*256+lonam#+ad
    namlen#:=PEEK(namstart)-4
    n$:=""
    FOR t#:=1 TO namlen# DO
      n$:+CHR$(PEEK(namstart+t#+3))
    ENDFOR t#
    RETURN n$
  ENDFUNC proc'name$
ENDPROC procs ■
```

Typing

by Henry Farkas

One of my daughter's school teachers asked me if I could help convert a typing tutor program. They bought one for the C64, but also needed to use it on a bunch of PETs. I told them that I didn't know anything about PETs, but that I'd write them a typing tutor in C64 BASIC and they could get someone to type it in on the PETs and save it to disk. After I did that, I decided to share it with the COMALites. I had just received *Today Disk #15*. It contained a bunch of procedures that enabled me to do the program in COMAL 0.14 with about the same ease that I find when programming in 2.0. *New'typing'tutor* program is on *Today Disk #18*.

I'd like to explain a bit about the philosophy of this program. When I was in junior high school back in the middle ages, the theory was that we should learn to type with very few mistakes. It took a long time to roll the paper up, get the eraser, erase, waft the little pieces of rubber away from the guts of the typewriter, and get back to the spot where you were typing. That shows you how old I am. They hadn't even invented snowpaque yet. Things are better now with word processors. So I decided that you really don't need all that much accuracy or speed in order to pass the course. The main thing is to get the idea well enough to type with 80% accuracy at 15 words/minute. After that, just typing whatever you want to type will increase your typing ability.

If you don't agree with this philosophy, you can change the parameters. If you list the **setup** procedure, you'll see that the first two numbers defined are called **acc'req** and **sp'req**. These stand for accuracy required and speed required. Just change them to any speed in words per minute and accuracy in percent that you want.

[*Editor's note: if you want a typing tutor for a Dvorak keyboard, try our typing tutor from the typing disk.*] ■

Extra Programs

1520 DIRECTORY PRINTER

This 0.14 program reads a disk directory and prints it in the 80 column mode of the 1520 plotter or normally to a printer. It prints the directory in multiple columns so that it can be pasted onto the front of a disk jacket (similar to the program we use here to create the directories with our HP LaserJet).

AMORTIZATION PROGRAM

This COMAL 0.14 program helps with loan calculations. Given the initial loan (principle), interest rate, and payback period (months) the payment is generated. A detailed history of payments and its reflection on the principle can also be printed.

GEOMETRY LESSON

This program teaches about circles and their properties. Covered are the diameter, radius, circumference and area of the circle (and how they are interrelated). PI, and its symbol, are also discussed. Graphics are used extensively to help teach the subject.

TWO DRIVE COPY PROGRAM

This 2.0 program will duplicate a disk and requires 2 disk drives. The copy is done by only copying the sectors on the disk that are used by files. This makes the copy go much faster. It formats your target disk before it starts copying to it, so make sure that it is a blank disk (or one you want to completely erase and re-use).

NEW FONTS

We have received several new fonts, and are including some of them on the disk. They will work with the COMAL 0.14 font viewer and the COMAL 2.0 font editor on the **Font Disk**, as well as with nearly any COMAL 2.0 program. ■

Spread Sheet



by Robert E. Webb

Spread sheet programs allow the user to generate a two dimensional array of numbers with labeled columns and rows. The numbers can be inserted directly into the array or be generated as a function of some other number in the array. Spread sheets are useful in that new numbers can be inserted in a "what if" mode and the entire spread sheet can then be recalculated using the same algorithms. This shows the effect of changing one parameter, on the many dependent variables.

Spread'sheet on *Today Disk #18* is a menu driven program which allows the user to format and label the rows and columns easily. It provides an address map of the array, allows the content to be manipulated by algorithms written in COMAL, and displays or prints the pre-formatted result. The COMAL algorithms are written by the user as an External Procedure, which is called from the *spread'sheet* program.

There are two menus, the main menu, and a format sub-menu. The following describes these two menus and each available option.

Main Menu Options

1) Help: explains how to use the program with step by step instructions. The heading for the external procedure is also shown.
2) Read disk file: loads a previously stored format and array into *spread'sheet*.
3) Edit format: Calls the **Edit** format menu.
4) Print map: Prints a map of the array showing the row and column names chosen by the user, and the cell addresses accessed by the external procedure.
5) Run external procedure: Executes the user's external procedure and stores the results.
6) Display spread sheet: Displays the spread sheet array, filled by option 5 above.
7) Write disk file: Stores the format and array on the disk.
8) Print spread sheet: Prints the spread sheet array on the line printer. Each column of numbers will be right justified and will occupy exactly 10 character positions.

Edit Format Menu Options

1) Edit title and size:
   a) Allows a title and subtitle of up to forty characters to be assigned to the spread sheet.
   b) Allows the number of rows and columns to be assigned, up to 28 rows and 8 columns maximum.
2) Edit row names: Allows names of up to 10 characters to be assigned to the rows.
3) Edit Column names: Allows column names of up to 10 characters to be assigned.
4) Edit decimal position: Allows a fixed decimal position to be assigned to each column. Values must be in the range of 0 to 6. Spread sheet numbers will have up to nine characters displayed or printed including the - sign and decimal point.

External Procedure

The form of the procedure is as follows:

```
PROC filename (REF ss(,)) CLOSED
  <<user code>>
ENDPROC filename
```

ss(,) is the spread sheet two dimensional array. Any cell in the array can be accessed with **ss(row,col)**. For instance the cell at row 5, column 7 can be set to 100 by the COMAL statement: **ss(5,7):=100**

Row names and column names can be assigned numbers so the cells can be addressed by name. The example above could be written as:

```
april:=5 // row name
payment:=7 // column name
ss(april,payment):=100
```

more»

Since the procedure is **CLOSED**, all variable names included are local to the procedure and are not visible to *spread'sheet*. Likewise no variable in *spread'sheet* can be accessed by the external procedure. Only the array ss(,) is common to both.

Demonstration Spread Sheets

Two demonstration spread sheet data files, *ssdemo1.dat* and *ssdemo2.dat* are provided on *Today Disk #18* so that the reader may more easily understand the use of *spread'sheet*. The *ssdemo1.dat* is a mortgage payment table showing payment, interest, principal, and principal balance for one year. The *ssdemo2.dat* is a sales and profit forecast for a product to be sold in many different regions.

These tips should be helpful to someone wanting to learn to use this spread sheet program:

1) **RUN "spread'sheet"**
2) Choose option 2 from the main menu to read a disk file.
3) You will see the disk directory scroll by. To load the mortgage data file, type: **ssdemo1** and press «*return*» twice.
4) Choose option 3 from the main menu to enter the Edit Format menu.
5) Choose option 1 from the Edit Format menu to see the title and size of the data file.
6) Press «*return*» to see but not change the original data. After all the fields have been displayed, you are returned to the Edit Format menu.
7) Choose the other options from the Edit Format menu to see other aspects of the data. Press «*return*» to return to this menu.
8) Choose item 0 from the Edit Format Menu to return to the main menu.
9) Choose item 0 from the main menu to quit.

You may also find it educational to list the external procedure for this file.
LOAD "ssdemo1.ext" and then **LIST**. ■

Super Chip



by Joe Potter

Circle80 is converted from the COMAL 0.14 circle procedure in *COMAL Today #15* page 44 to draw a circle on the hi-res RGB display of the C128 with Super Chip. Just use it like this:

```
USE c128
setgraphic80
circle80(xpos,ypos,radius)
```

Be sure to flip your monitor to the 80 column display to see the circle. The program below draws concentric circles.

```
USE c128
turbo(TRUE)
setgraphic80
FOR radius:=20 TO 160 STEP 20 DO
  circle80(320,100,radius)
ENDFOR radius
turbo(FALSE)
//
PROC circle80(x,y,rad) CLOSED
  USE c128
  y':=0; y'offset:=1.75; start:=TRUE
  TRAP
    FOR i#:=1 TO 64 DO
      temp:=rad*.995004165-y'*.0998334166
      y':=y'*.995004165+rad*.0998334166
      rad:=temp
      sx:=1.4*rad+x
      sy:=y-y'/y'offset
      IF start THEN
        plot80(sx,sy)
        old'x:=sx; old'y:=sy; start:=FALSE
      ELSE
        line80(old'x,old'y,sx,sy)
        old'x:=sx; old'y:=sy
      ENDIF
    ENDFOR i#
  HANDLER
    settext80
    print80(12,10,"Values out of range")
  ENDTRAP
ENDPROC circle80 ■
```

De-Link Package 

by Dick Klingens, Dutch COMAL Users Group

You have done a lot of typing: 500+ lines are in computer memory. A new program is born. The only thing left to do is to link a package onto it and then test the program. The package name is *pkg.text*; but by mistake you typed:

LINK "pkg.test" *// oops!*

And that package is rommed. What to do? **DISCARD** does not help you: a rommed package cannot be discarded. There are some possibilities:

1. Switch the computer off and on and ... I don't like this method. Do you?
2. **LIST "your program"** *// to disk*
   SYS $c87b *// cold start of COMAL*
   ENTER "your program"
3. or type the following lines
   MERGE "proc.ramall" *// if on your disk*
   SCAN
   ramall
   DEL ramall
   DISCARD

Now type:

LINK "pkg.text" *// the right one!*

Method 3 is a time saving one. It is based on the small procedure at the end of this article.

It is possible to have a total of ten libraries (groups of packages) in memory at a time (8 made by yourself and 2 standard *[Super Chip adds two more libraries]*). The number of stored libraries is in address $c7ef. Starting from $c7f0 we find the low byte of the start address of each package; the high bytes are located from $c7fa. The package pages are stored from $c804. The package itself contains its *rommed* status. *COMAL 2.0 Packages* by Jesse Knight explains how packages are organized. For example, here is a package header.

```
.lib c64symb
*=<start address>
.byte <page>+rommed
```

From the package header we deduce that *to be rommed or not* is determined by the first byte. The value of **ROMMED** is added to the page. It equals %00010000. E.g. if **DEFPAG** was used as the page for the package, we have:

```
DEFPAG  = %01000110
ROMMED = %00010000
         %01010110
```

To filter the rom bit, use **BITAND**:

```
PAGE = %01010110
bitand  %11101111 (mask)
        %01000110
```

This process changes a rommed package into a rammed one which can be discarded, but leaves an already rammed package alone.

```
PROC ramall CLOSED
  USE system
  libpt:=$c7ef // number of libraries
  liblo:=$c7f0 // lo start each lib
  libhi:=$c7fa // hi start each lib
  libpag:=$c804 // page of each lib
  mask:=%11101111
  number:=PEEK(libpt)
  FOR i#:=1 TO number DO
    lo:=PEEK(liblo+i#-1)
    hi:=PEEK(libhi+i#-1)
    pag:=PEEK(libpag+i#-1)
    setpage(pag)
    ram(lo+256*hi)
  ENDFOR i#
  END "All libraries are rammed now"
  //
  PROC ram(adrs)
    POKE adrs, PEEK(adrs) BITAND mask
  ENDPROC ram
ENDPROC ramall ■
```

Stacks Revisited

by Richard Bain 

I discussed scope rules and stacks in *COMAL Today #14* and *#16*. This issue discusses the motivation of a two stack system and continues with details on what information is stacked for the various COMAL 2.0 structures on the C64.

Do all computer languages support a two stack system? No, BASIC and many other languages find one stack sufficient. As you will see, with two stacks, COMAL has overcome the problems created by one stack systems.

When you assign some characters to a string in BASIC, they are stored in the data area. The string only occupies as much memory as is required for its length. Later, you might add to the string. BASIC is not smart enough to reserve the extra space needed to do this. It must move the entire string to the end of the data section. This leaves a gap where the original string was stored. When BASIC runs out of memory, it reclaims lost space by sliding variables around to fill the holes. This slow process is called garbage collection.

COMAL requires you to dimension strings before using them. (C64 COMAL 2.0 dimensions strings to 40 characters if you don't dimension them yourself.) Because of this, the exact space needed for the string is reserved; COMAL never needs to move a string to a bigger spot. This explains why COMAL does not have garbage collections.

Another feature of COMAL is the ability to use temporary variables (such as parameters), and local variables in **CLOSED procedures** and **FOR** loops. These variables only take up space for a short time; the memory is reclaimed by COMAL as soon as the loop ends or the **procedure** is exited.

Before I can explain why COMAL needs a second stack (or data area if you prefer) to store its variables, I must explain the concept of local variables. Most variables in COMAL are global. This means they may be used anywhere in the program. There are three exceptions to this rule: variables in **CLOSED procedures**, parameters (in both open and **CLOSED procedures**), and the counter used in the **FOR** loop.

All variables used in a **CLOSED procedure**, except those which were imported into the **procedure**, are local. This means that the **procedure** has no way to know about variables used outside the **procedure**. It also means that once the **procedure** is exited, its variables are forgotten. The memory those variables had occupied will be available for the rest of the program to use as necessary.

Parameters are a complicated form of local variables, but the same rules apply. The parameters may be used from within the **procedure**, but the memory they require will be reclaimed by COMAL as soon as the **procedure** is exited.

The **FOR** loop counter is also local. Memory is allocated for this special variable when the loop is entered. The same memory is deallocated when the loop is exited.

COMAL supports several types of temporary variables. Memory must be allocated for these variables and later reclaimed. **In the same spirit of efficient memory management used with strings, COMAL requires that there will be no gaps in any memory it uses.** A stack is a tool that supports this type of memory management. However, there is a catch using stacks. The data most recently placed on the stack must also be the first data to be removed. This means that a variable which will last a long time cannot be placed on the stack on top of a variable that only lasts a short time. We will discuss how COMAL operates under this

more»

restriction by studying how long some variables last in relation to others.

Global Variables

The easiest type of variable to study is the global variable. Once it is created, it remains in memory during the rest of program execution (possibly longer). A global variable cannot be placed on the stack on top of a local variable. Otherwise, a hole would be left in the stack when the local variable was no longer needed. There are two common ways to obey this rule: reserve memory on the stack for all global variables at the start of the program, or separate local and global variables. Pascal uses the first method, COMAL uses the second. (BASIC does not have local variables, so the rule does not apply.) Now you know **why** COMAL uses two stacks, one is for short lived data, and the other is for long lived data. However, more study is required for long lasting, but not global, data.

Using a one stack system, any variable can be placed on the stack as long as two conditions are fulfilled. The variable must remain active at least as long as any variable placed on top of it, but not longer than any variable stacked before it. Using two stacks, the same rules must be followed, but having a choice of stacks greatly increases flexibility. A variable placed on one stack **can** remain on its stack longer than a short lived variable previously placed on the **other** stack. This is exactly what COMAL does. Variables that last a long time generally go on the upper stack and short lived variables generally go on the lower stack. The only idea that still needs to be clarified is how long each kind of variable lasts.

Local Variables

Let's examine the local variable created in a **CLOSED procedure**. The important questions to ask are *will this variable last longer than any variable created before it* and *will this variable last as long as any variable created later.* If it lasts beyond the life of a previous variable placed on the lower stack, it must go on the upper stack. If it doesn't last as long as a future variable it must go on the lower stack. If both conditions hold true, there is a problem.

It turns out that it is safe to put the variable on the upper stack. COMAL does not allow you to create a global variable from within a **CLOSED procedure** so when the **procedure** is exited, all its variables will be destroyed. This happens before any new global variables can be created. Notice how a simplified model of a stack will respond to the following program

```
0010 a:=5
0020 EXEC p
0030 b:=a
0040 //
0050 PROC p CLOSED
0060   c:=10
0070 ENDPROC p
```

Program execution flows from line 10 to line 60 to line 30. This is how the upper stack will look after each line (the variable name is placed next to its value below):

line	upper stack
10	5 (*a*)
60	5 (*a*)
	10 (*c*)
30	5 (*a*)
	5 (*b*)

The global variable, **a**, is placed on the stack from line 10. Then **procedure p** is entered and the local variable, **c**, is placed on the upper stack from line 60. In the illustration above, the stack grows downward. Therefore, **c** is *on top* of **a** on the stack even though it is below **a** in illustration. Then the procedure is exited and **c** is no longer needed. This makes it possible

more»

for b, created in line 30, to occupy the **same** location on the stack that c used to occupy.

FOR Loop Counter

The next variable we will discuss is the counter in the **FOR** loop. This variable is destroyed as soon as the loop ends, so it cannot be stored on the upper stack with the global variables, can it? Remember, local variables from a **CLOSED procedure** are destroyed when the **procedure** ends and they can still go on the upper stack. The difference is that global variables cannot be created within a **CLOSED procedure**, but global variables can be created from within a **FOR** loop. Therefore, the counter must go on the lower stack. The following program requires both stacks.

```
0010 a:=5
0020 FOR x:=1 TO 2 DO
0030   b:=a
0040   c:=a+b
0050 ENDFOR x
0060 PRINT c
```

Program execution flows from line 10 to lines 20, 30, 40, 30, 40, and 60. Both stacks are shown below (lines 30 and 40 are only show during the first pass).

line	lower stack	upper stack
10		5 (*a*)
20	1 (*x*)	5 (*a*)
30	1 (*x*)	5 (*a*)
		5 (*b*)
40	1 (*x*)	5 (*a*)
		5 (*b*)
		10 (*c*)
60		5 (*a*)
		5 (*b*)
		10 (*c*)

The order of the variables created above is: a, x, b, c. During execution of line 40 above, all four variables are active at once. Because x was separated from the other variables on the upper stack, it was possible to remove x from program memory without creating a gap between variables a and b.

Parameters

The last type of variable to be discussed is the parameter. Parameters from **CLOSED procedures** can be treated just like the local variables created inside the **procedure**. This is not the case for open **procedures**. In an open **procedure**, the parameters will be destroyed, but it is possible to create global variables from within the **procedure** that will remain when the **procedure** ends. Therefore the parameters from open **procedures** (and **CLOSED procedures** as well) are stored on the lower stack. Imported variables are treated exactly like **REF** parameters, they have the same bits set in the name table entry as described in *COMAL Today #16*, and the same bytes are used in the tokenized COMAL code to use the variables. However, imported variables use the upper stack even though parameters use the lower stack.

Stack Entries

This completes the discussion on how variables are stored on the stacks. Other items are placed on the stack as well. System variables must be saved during the following structures: **FOR** loop, **TRAP .. HANDLER** structure, **procedures** and **functions**. (There are no stack entries for the **LOOP**, **REPEAT** and **WHILE** loops, or the **CASE** and **IF** structures.) The following chart summarizes how the information is stored. The information for these stack entries is pointed to by FORPT, a system variable located at addresses 16 and 17. A brief description is given to help the curious recognize what they will see if they look at the COMAL stacks. Finally, short routines will be given to show how this information can be used.

more»

FOR LOOP - Real Variable	
bytes	meaning
2	old FORPT
1	0 for FOR loop - real
2	name table entry address
3	old name table entry
5	loop counter
5	ENDFOR value
5	STEP value

FOR LOOP - Integer Variable	
bytes	meaning
2	old FORPT
1	1 for FOR loop - integer
2	name table entry address
3	old name table entry
2	loop counter
2	ENDFOR value
2	STEP value

TRAP (TRAP / HANDLER)	
bytes	meaning
2	old FORPT
1	2 for TRAP
2	address: TRAP statement



Get the facts and fear no more!

HANDLER (TRAP / HANDLER)	
bytes	meaning
2	old FORPT
1	3 for HANDLER
2	address: REPORT statement
2	ERR (hi/lo)
1	ERRFILE
1	LEN (ERRTEXT$)
LEN	ERRTEXT$

PROC or FUNC	
bytes	meaning
2	old FORPT
1	12 for PROC or FUNC
1	type (see below)
2	address: calling line
1	offset into calling line
1	number of parameters
2	old LOCLPT
5*	3 byte name table entry 2 byte address to above
	*repeat for each parameter
more	the parameter values go here

Type Byte for PROCs and FUNCs		
	Open	CLOSED
PROC	0	16
FUNC str	3	19
FUNC real	4	20
FUNC int	5	21

more»

The preceding page contains the templates for system information which are stacked each time a structure is entered. Each template should be considered as a single unit or data structure which is placed on the lower stack as the structure is entered. It is removed from the stack when the structure is exited.

All the entries start with the same format: two bytes for the pointer to the previous stack entry (one of the templates described above) and one byte to specify the type of stack entry. A zero in the high byte of the pointer indicates that there is no previous stack entry. *COMAL Today #16* explains that FORPT (addresses 16 and 17) points to the start of the current stack entry on the lower stack.

The **FOR** loop entry follows these three bytes with information necessary to make the counter a **local** variable. It saves the three bytes from the variable's original name table entry (these bytes tell if the variable was a string or number etc., and where it was located). It also saves the address where to restore the name table entry when the loop ends. Next, it reserves room for the counter, the final value used to stop the loop, and the step value. Then it overwrites the counter's name table entry indicating a number located at the ninth byte of this stack entry. When the loop terminates, the variable replaced by the counter is restored and the stack entry is destroyed.

The **TRAP** structure only adds the address of the **TRAP** statement to the first three bytes of the structure entry. The address of the corresponding **HANDLER** statement is coded into the **TRAP** line. When an error occurs, COMAL can check the stack entry for the address of the **TRAP** statement, and follow the pointer in this program line to the **HANDLER** section.

Suppose the error occurred inside a **FOR** loop. When COMAL checks the third byte of the top stack entry, it will see that is a **FOR** structure and not a **TRAP** structure. COMAL must first terminate the **FOR** loop and restore the variable which was replaced by the counter. It can then follow the first two bytes of the **FOR** loop stack entry to find the stack entry for **TRAP**. Then these two entries can be popped off the stack and the **HANDLER** entry can then be created.

The **HANDLER** entry places the common three bytes on the stack and then information relating to the error. It places the address of the line where the error occurred on the stack followed by the error number, the file number involved, the length of the text for the error message, and the error text itself.

The **procedure** and **function** stack entries are identical to each other. They start with the three bytes for the previous stack entry link and the stack entry type just as for the other structures. The next byte distinguishes **procedures** from **functions** and tells whether or not they are closed. The next two bytes give the address of the calling line to tell COMAL where to go after the **procedure** ends. The following byte is the offset into the calling line, making it possible to include more than one **procedure** call on a single line. Then the number of parameters is given. The value from LOCLPT (addresses 14 and 15) is stacked to help maintain proper scope. (In **CLOSED procedures**, the values of SCLSD1 and SCLSD2 are placed on the upper stack.) Then five bytes per parameter are stacked in a manner identical to how the counter for the **FOR** loop is handled. Finally, the values of the parameters are stacked and their name table entries are updated to reflect this.

When the **procedure** ends, it is a simple matter to restore the variables which were replaced by the parameters. Just go to the eighth byte of the stack entry to find the number of parameters. Then for each parameter, look up the address of the name table entry. The

more»

address for the first parameter is found in the 14th and 15th bytes of the stack entry. Then copy the three bytes for the name table entry from the 11th, 12th, and 13th bytes of the stack entry. Once the parameters are restored, the stack entry is deleted.

CLOSED PROC/FUNC variable info	
bytes	meaning
3	original name table entry
2	name table entry address
2	link to next variable

The last stack entry to be discussed is not a structure entry. It is the information placed on the upper stack when a variable is used for the first time within a **CLOSED procedure** or **function**. Since this variable is local, information for the global variable it replaces must be saved. As with the counter in the **FOR** loop and with parameters, five bytes must be saved (three bytes for the name table entry and two more for the address of that entry). Two more bytes are also stacked. These bytes are links in a chain for all the local variables created within the **procedure**. They contain the address which is one byte less than the start of the seven bytes stacked for the previous entry. The start of the chain is pointed to by LOCLPT (addresses 14 and 15). The end of the chain is indicated by a zero in the high byte of the link. This chain makes it fast and easy to restore the global variables upon exiting a **CLOSED procedure**.

The information presented here has only been of theoretical interest, but it can be put to practical use. The *tron* program in *COMAL Today #15* used this information to print line numbers as each line was executed. The **print'line'number procedure**, listed below, prints the line number of the line which called it. It first finds the stack entry pointed to by addresses 16 and 17. It then looks at the fifth and sixth bytes to find the address of the calling line. The first two bytes of that line contain the line number plus 10000.

```
PROC print'line'number CLOSED
  pt:=PEEK($10)+256*PEEK($11)
  la:=PEEK(pt+4)+256*PEEK(pt+5)
  ln:=256*PEEK(la)+PEEK(la+1)-10000
  PRINT USING "####": ln
ENDPROC print'line'number
```

Another use of this information is to find where variables are stored in memory. Someone wishing to learn about how real numbers are stored can first find out where the variable is located. Then assign a value to the variable and **PEEK** the memory. When you think you know what you are doing, try poking the memory and printing the value of the variable.

```
FUNC find'real (REF num) CLOSED
  t:=PEEK($10)+256*PEEK($11)
  t:=PEEK(t+13)+256*PEEK(t+14)
  t:=PEEK(t+2)+256*PEEK(t+3)
  RETURN PEEK(t)+256*PEEK(t+1)
ENDFUNC find'real
```

The first assignment above sets t to the start of the **function's** stack entry. The next line sets t to the address of num's name table entry. It is important that num is a **REF** parameter so it will have a value at the same location as variable in the main program you are trying to find. *COMAL Today #16* explained that the third and fourth bytes of the name table entry point to value of the variable. In the case of a **REF** parameter, this value is actually an indirect pointer to the address we are looking for. The third assignment above finds the address from the name table entry and the value returned by the **function** is our desired address.

Another use is storing a simple machine language routine in protected memory. Using

more»

the cassette buffer is risky because it is used by several routines (including some from Super Chip) which think it is a safe place. However, if you dimension a string in COMAL, you can be sure that the string will reserve a safe spot for your private use. Then you can place relocatable machine code in the string and SYS directly to it. All you have to do is calculate where the string starts. The following program uses this technique to change the border color of the C64 text screen.

```
DIM a$ of 4
FOR x:=1 to 4 DO
  READ t
  a$(x):=CHR$(t)
ENDFOR x
FOR x:= 1 TO 16 DO SYS find'str(a$)+4
//
DATA $ee,$20,$d0 // inc $d020 ;border color
DATA $60 // rts
//
FUNC find'str (REF s$) CLOSED
  t:=PEEK($10)+256*PEEK($11)
  t:=PEEK(t+13)+256*PEEK(t+14)
  t:=PEEK(t+2)+256*PEEK(t+3)
  RETURN PEEK(t)+256*PEEK(t+3)
ENDFUNC find'str
```

The **function** find'str is identical to the **function** find'real except for the parameter in the header, and that parameter isn't even used. Well, actually the parameter is used. That is what this entire article is all about. The **functions** must be different to avoid a *wrong parameter type* error. Note: the +4 in the SYS command above compensates for the four bytes a string uses to remember the dimensioned and current lengths.

This concludes my series on stacks. Perhaps someone will want to dig further into the subject for the shortest lived stack structures, intermediate results from calculations and the PRINT statement. If so, I would like to hear from you. ■

XPL/0



by Dick Klingens, Dutch COMAL Users Group

The language XPL/0 is an extension of the PL/0 language created by professor Niklaus Wirth. It serves as an example language in a chapter on compiler techniques in his book *Algorithms+Data Structures=Programs*. In this book professor Wirth develops a compiler system capable of compiling and executing PL/0 programs. This system is implemented by one program written the high level language Pascal.

A COMAL implementation of the XPL/0 system is now available on Holland Disk #9. Yes, this program shows how you can write a new language in COMAL. The program has been modularized in order to deal with the storage restrictions of an interactive environment such as the one offered by COMAL.

It must be emphasized that neither the language XPL/0 nor its COMAL based implementation are intended for serious use by programmers. It is an instructive example of the forces of COMAL and of compiler techniques.

If you are interested in how languages can be created, get professor Wirth's book and *Holland Disk #9*. A text file is included on the disk that briefly explains the XPL/0 system.

[This is just one of the things on Holland Disk #9. It includes many more good programs. We are getting so many good programs from Europe, that we have released them as a set of 12 disks. We think that the programs are very good. You can see for yourself. Just order our European 12 Disk Set.

Even though most of these programs are not mentioned in COMAL Today, they are well documented. Most of the European disks include the Read and Run system (see COMAL Today #16) for easy access to the text files. Super Chip can also be used to type the "mini COMAL newsletter".] ■

Apple Notes

Progress on Apple COMAL has been good since the last *COMAL Today*. I now have a preliminary version I am sending to people who have purchased version 1.0. For those people each issue of *Apple COMAL Notes*, starting with issue #4 (current issue is #5), will come with the latest version of Apple COMAL.

The following is a list of working keywords in preliminary Apple COMAL version 0.39. Additional commands are not fully implemented yet.

VERSION 0.39 KEYWORDS

ABS	EOD	PEEK
AND	EPSILON	PI
APPEND	ESC	POKE
AUTO	EXEC	PRINT
BASIC	EXP	PROC
BYE	FALSE	READ
CASE	FILE	RENUM
CHR$	FOR	REPEAT
CLOSE	FREE	RESTORE
CLOSED	FUNC	RETURN
CON	GOTO	RND
COS	IF	RUN
CURCOL	IN	SGN
CURROW	INKEY$	SIN
DATA	INT	SIZE
DEL	INPUT	SPC$
DELETE	LABEL	SQR
DIM	LEN	STEP
DISPLAY	LET	STOP
DIV	LIST	TAB
DO	LOG	TAN
EDIT	MOD	THEN
ELIF	NEXT	TO
ELSE	NOT	TRAP ESC
END	NULL	TRUE
ENDCASE	OF	UNTIL
ENDFUNC	OPEN	WHEN
ENDIF	OR	WHILE
ENDPROC	ORD	WRITE
ENDWHILE	OTHERWISE	ZONE
ENTER	PAGE	

COMPATIBILITY

I am trying to make Apple COMAL as compatible with other 2.0 COMAL's as I can. The only features I do not plan on implementing for version 1.0 are packages and external procedures. I am using the same device notation (ds: for screen, lp: for printer, etc) as CP/M COMAL and C64 COMAL.

NEW FEATURE

While maintaining compatibility, I also am implementing some new ideas and concepts to help the COMAL programmer. One of these ideas is a numeric/string operator to test for similarity, not just an exact match. The operator is == and is pronounced *is similar to.*

For strings this operator compares the strings regardless of case. For numbers, the comparison checks for zero and if necessary tests:

ABS((num1-num2)/num2)<=EPSILON

EPSILON is both a function and a command (like **ZONE**). The default value is 0.0000001, but you can set it to any value you wish. This operator is very useful to handle floating point rounding problems. For example, 13/3*3=13 is false due to floating point error. 13/3*3==13 is true because the numbers are similar.

OFFER ENDING SOON

When version 1.0 is released I will no longer accept subscriptions to *Apple COMAL Notes*, nor will I sell copies as a retailer. If you wish to purchase Apple COMAL now, receive *Apple COMAL Notes* and save money too, please write or call for further details.

David Stidolph, COMALites United
1670 Simpson #102
Madison, WI 53713
(608)222-4505

Disk Sleeve Directories

contributors:

Bob Bain
Richard Bain
Marty Balash
Jack Baldridge
Ora L Flininghamm
Gerald Hobart
Bill Inhelder
Len Lindsay
Sid Sieferlein
Mark Skopinsky
David Stidolph
Paul Van Leeuwen

Today Disk #17 - Front

			130 Files	6 Blocks Free:
boot c64 comal	fourier'series	epson.exp'italic	----------	imag.dog'left
slowboot	guess'number	epson.italic	- data file -	imag.dog'right
c64 comal 0.14	hopping	epson.nlq	----------	imag.fish'left
comalerrors	invisible	epson.permexpand	screenhelp.obj	imag.fish'right
ml.fastload-msd	polyfit	epson.pica	----------	imag.girl'left
ml.sizzle	powers'of'2	epson.printinfo	- sprite image -	imag.girl'right
hi	pressure	epson.proportion	- files -	imag.plane'left
menu	russian'roulette	epson.sub/super	----------	imag.plane'right
window-rs232.obj	sample'rate'calc	epson.underlined	imag.'a'	imag.queen
----------	sprite'editor62	----------	imag.'c'	imag.rhianon
- comal 0.14 -	subtract	-graphics demos-	imag.'f'	imag.sun
- programs -	target	----------	imag.'k'	imag.target
----------	target'sighting	tv'commercials	imag.'l'	imag.truck'left
add'practice	test'precision	tv.marquee+	imag.'m'	imag.truck'right
barometer	test'scope	tv.bang'bang	imag.'n'	----------
box	voting'game	tv.cue'card	imag.'o'	-for more info -
calculate'e	word'hider	tv.second'act	imag.'r'	- send sase to -
calculate'pi	----------	tv.finish	imag.bird'left	----------
crystal'ball	- epson -	----------	imag.bird'right	- comal users -
define'shape	-printer demos -	- function -	imag.boat'left	- group, usa -
demo/1750	----------	----------	imag.boat'right	- 6041 monona -
demo/line'length	epson.condensed	peek'hires.func	imag.boy'left	- madison, wi -
demo/screenhelp	epson.doublestrk	----------	imag.boy'right	- 53716 -
direction'game	epson.elite	- procedure -	imag.car'left	----------
envelope'artist	epson.emphasize	----------	imag.car'right	-(608)222-4432 -
expression	epson.expanded	line'length.proc	imag.comal today	----------

contributors:

Bob Bain
Richard Bain
Jack Baldridge
Marcel Bokhorst
Bob Hoerter
Dick Klingens
Len Lindsay
Bob McCauley
Bill Nissley
Mark Skopinsky
David Stidolph
Sister Anne Stremlau
Paul Van Leeuwen

Today Disk #17 - Back

			76 Files	2 Blocks Free:
hi	read'printertext	ext.directions	src.gemini	----------
----------	russian'roulette	ext.enter'record	src.printer	-for more info -
- comal 2.0 -	test'precision	ext.split'file	----------	- send sase to -
- programs -	test'scope	ext.sub-file	- packages -	----------
----------	set'cursor	----------	----------	- comal users -
antivoter	spin&win'tools	- procedures -	pkg.cleanup	- group, usa -
barometer	spin&win'writer	----------	pkg.gemini	- 6041 monona -
batch'to'package	spin'and'win	proc.nofont	----------	- madison, wi -
calculate'pi	subtract	proc.print1char	- data files -	- 53716 -
chip.read'text	videofilersystem	proc.set	----------	----------
crystal'ball	voting'game	proc.to'file	bat.set'font	-(608)222-4432 -
demo/1750	----------	proc.tran	dat.main'video	----------
draw poker	- external -	----------	dat.states	
kernal'tutorial	- procedures -	- source code -	hrg.poker	
makescreen	----------	----------	scrn.makehelp	
pressure	ext.correct'file	src.customdump	txt.printer pack	

Holland Disk #2

			87 Files	95 Blocks Free:
hi	----------	src.calc	- basic boot -	dat.name
menu	- external -	src.mccomal	- program for -	help-text
----------	- procedure -	src.text	-enhanced 0.14 -	hrg.field
- comal 2.0 -	----------	----------	----------	----------
- programs -	ext.name	- packages -	boot'ex'comal	- dutch comal -
----------	----------	----------	----------	- users group -
demo/calc	-listed program-	pkg.calc	- text files -	----------
demo/dualscreen	----------	pkg.mccomal	----------	-dick klingens -
demo/mccomal-1	lst.delink	pkg.text	txt.enhancer	-violier 3 -
demo/mccomal-2	----------	----------	txt.bootusa	-krimpen ad -
demo/sets	- procedures -	- comal 0.14 -	txt.delink	- yssel-
demo/text-1	----------	- files -	txt.help package	-2925 te -
demo/text-2	proc.fontex	----------	txt.machine lang	-the -
dmon rev 1.3	proc.sets32	- procedures -	txt.sets	-netherlands -
flash'cursor	proc.soundex	----------	----------	----------
labelmon	----------	beep.proc	- data files -	
pigeons	- source code -	fkeys.proc	----------	
soundeffect	----------	----------	bin.dualscreen	

Holland Disk #1

105 Files — 33 Blocks Free:

hi	wa'kans2	proc.fig8	shape.li wim	- listed -
----------	wa'kansverd	proc.fig9	shape.ne wim	- programs -
- comal 2.0 -	wim	proc.nolist	shape.op wim	----------
- programs -	wim's werk	proc.piep	shape.re wim	- use enter to -
----------	----------	proc.scrolldown	----------	- merge lines -
comal'demo	- external -	proc.sirene	- dmon -	----------
crossref	- procedure -	----------	- help files -	envelope.l
dmon	----------	- source code -	----------	generate'error.l
door's close up	ext.name	----------	txt.help	use'sound.l
error	----------	src.basic	txt.help.dutch	----------
fonteditor	- procedures -	src.dutch	----------	- dutch comal -
listen'to'tape	----------	src.tape	- data files -	- users group -
logoles	proc.extnd'sound	----------	----------	----------
logopen	proc.extd'graph	- packages -	comalerrors	-dick klingens -
make'err'dutch	proc.fig1	----------	dat.name	-violier 3 -
make'err'file	proc.fig2	pkg.basic	----------	-krimpen ad -
make'hex'file	proc.fig3	pkg.dutch	-character font-	- yssel-
run'basic'prog	proc.fig4	pkg.tape	----------	-2925 te -
testdisk20	proc.fig5	----------	font.thin chars	-the -
wa'dobbelen	proc.fig6	-sprite shapes -	----------	-netherlands -
wa'kans1	proc.fig7	----------	- comal 0.14 -	----------

Holland Disk #3

60 Files — 249 Blocks Free:

hi	----------	----------	txt.letter	----------
menu	- source code -	- comal 0.14 -	txt.no basic	-dutch comal -
----------	----------	- enhancer -	txt.packages	- users group -
- comal 2.0 -	src.arrayfnc	----------	txt.random fix	----------
- programs -	src.codeload	bootm	txt.reprograming	-dick klingens -
----------	src.notbasic	comal.14	txt.run protect	-violier 3 -
demo/arrayfunc	----------	comalerrors	txt.sorting	-krimpen ad -
demo/codeload	- packages -	generror.n.l	txt.wp in comal	- yssel-
demo/notbasic	----------	----------	----------	-2925 te -
secret	pkg.arrayfnc	- text files -	- data files -	-the -
selfreprogram	pkg.codeload	----------	----------	-netherlands -
tekst	pkg.notbasic	txt.info-bootm	help-tekst	----------

Holland Disk #4

57 Files — 34 Blocks Free:

hi	gen rabmsg	pkg.rabbit	val.proc	----------
menu	reorg'dir	----------	val/str.lst	-dick klingens -
----------	----------	- comal 0.14 -	----------	-violier 3 -
- comal 2.0 -	- source code -	- enhancer -	- text files -	-krimpen ad -
- programs -	----------	----------	----------	- yssel-
----------	src.basic2	bootm	txt.bootm	-2925 te -
0.14	src.rabbit	comal.14	txt.double dutch	-the -
change comal	rabmsg	comalerrors	txt.letter	-netherlands -
demo/rabbit-1	----------	----------	txt.xcomal text	----------
demo/rabbit-2	- packages -	- procedures -	----------	
disassem20	----------	----------	- dutch comal -	
dutch comal	pkg.basic2	str.proc	- users group -	

Holland Disk #5

81 Files — 10 Blocks Free:

hi	----------	- libraries -	----------	txt.no pps
menu	- source code -	----------	rotate'help'1	----------
----------	----------	lib.labels	rotate'help'2	-dutch comal -
- comal 2.0 -	src.add	lib.tabel80	rotate'help'3	-users group -
- programs -	src.irq	symbols	----------	----------
----------	src.kol80	----------	- sample -	-dick klingens -
compiler-prelim	src.no'pps	-package demos -	- procedures -	-violier 3 -
demo-compiler	src.solve	----------	----------	-krimpen ad -
kernal/comal err	----------	demo/irq1	merge.add	- yssel-
popover(super)	- packages -	demo/irq2	merge.faculty	-2925 te -
read seq'wrdfil	----------	demo/irq3	merge.solve	-the -
rotate	pkg.add	demo/irq4	----------	-netherlands -
type quick	pkg.irq	demo/irq5	- text files -	----------
----------	pkg.kol80	demo/kol80	----------	
-listed program-	pkg.no'pps	demo/no'pps	txt.compiler	
----------	pkg.solve	----------	txt.irq package	
lst.compiler	----------	- data files -	txt.kol80	

Amiga COMAL Note

Just as we were ready to take this issue to the printers, we got a call.

It seems that the Amiga COMAL is running smoothly and can have separate windows. Depending on the number of windows used, the free user memory on the preliminary Amiga COMAL is:

150K-185K on Amiga 500

150K-185K on Amiga 1000

350K-400K on Amiga 1000 (memory expanded)

350K-400K on Amiga 2000

COMAL recognizes the full memory expansion, so as you add 500K to your Amiga, the COMAL can use it fully.

CLASSIFIED ADS

For sale: 4 Commodore CBM model 8032 computers, each with Instrutek COMAL boards version 2.0 with 128K. Also 4 CBM model 8250 dual floppy disk drives 1 Commodore 2022 Printer, 3 4023 printers and 2 8032 P printers. Best offer. Phone: (613) 332-3182 or (613) 332-1459 evenings.

For sale: 2 Commodore 4040 dual disk drives (working) and 2 Commodore 8250 dual disk drives (almost working) Best offer. Contact COMAL Users Group USA Limited, 6041 Monona Drive, Madison, WI 53716, phone: (608) 222-4432.

For sale: 1 McPen Light Pen, new! $29. Contact COMAL Users Group USA Limited, (608) 222-4432.

Holland Disk #6

100 Files — 5 Blocks Free:

hi	----------	----------	txt.real numbers	chaos0.14/3
menu	- function -	- packages -	txt.scroll left	chaos0.14/4
----------	----------	----------	----------	char'in'0.14
- comal 2.0 -	func.choice	pkg.irq0	- comal 0.14 -	----------
- programs -	----------	pkg.notbasic	- system -	- procedures -
----------	- procedures -	pkg.splitscleft	----------	----------
borge	----------	pkg.test	boot c64 comal	choice.proc
cave-warrior	proc.copyscreen	----------	slowboot	use'sound.proc
info-chaos	proc.nofont	-character font-	c64 comal 0.14	----------
chaos1	proc.numbers	----------	comalerrors	- dutch comal -
chaos2	proc.ramall	font.xround	ml.fastload-msd	- users group -
chaos3	proc.ram/rom/lib	----------	ml.sizzle	----------
chaos4	----------	- text files -	----------	-dick klingens -
char'by'char	- source code -	----------	- comal 0.14 -	-violier 3 -
demo/copyscreen	----------	txt.borge	- programs -	-krimpen ad -
demo/irq0	src.irq0	txt.char by char	----------	- yssel-
demo/numbers	src.notbasic	txt.copyscreen	go	-2925 te -
demo/splitscroll	src.splitscleft	txt.discard	info-chaos0.14	-the -
remove'libs	src.test	txt.discard font	chaos0.14/1	-netherlands -
textsprites	symbols	txt.letter	chaos0.14/2	----------

Holland Disk #7

72 Files — 224 Blocks Free:

hi	twin-drivecopy	dat.blck screen1	proc.add'helps	----------
menu	----------	dat.blck screen2	proc.bdir	- dutch comal -
----------	- source code -	dat.gen screen	proc.help	- users group -
- comal 2.0 -	----------	dat.keywords	proc.wblock	----------
- programs -	src.echo	viz.comalkey	----------	-dick klingens -
----------	src.text vs 1.02	----------	- text files -	-violier 3 -
bdir demo	symbols	- functions -	----------	-krimpen ad -
block-editor	----------	----------	txt.bdir	- yssel-
build-screens	- packages -	func.getran	txt.command	-2925 te -
command demo	----------	func.pa	txt.echo	-the -
demo/echo	pkg.echo	func.rblock	txt.letter	-netherlands -
demo/text	pkg.text	----------	txt.pitfall	----------
read viza-file	----------	- procedures -	txt.screens	
show-screens	- data files -	----------	txt.small recs	
small records	----------	proc.1541command	txt.twin drive	

Holland Disk #8

81 Files — 23 Blocks Free:

hi	set'cursor	ext.ghch.01	scrn.start	txt.simple logic
menu	simple-max	----------	----------	----------
----------	----------	- function -	- comal 0.14 -	- dutch comal -
- comal 2.0 -	- source code -	----------	- programs -	- users group -
- programs -	----------	func.max	----------	----------
----------	src.clock	----------	baro&mano(0.14)	-dick klingens -
3-d	src.text1.02	- procedures -	pressure(0.14)	-violier 3 -
baro&mano	symbols	----------	----------	-krimpen ad -
demo/clock	----------	proc.names	- text files -	- yssel-
demo/names	- packages -	proc.nonlist	----------	-2925 te -
demo/start	----------	----------	txt.3d drawing	-the -
flip	pkg.clock	- data files -	txt.baro&mano	-netherlands -
gambler	pkg.text	----------	txt.cursor	----------
ghosts	----------	dat.ghost	txt.letter	
info'csg	- external -	hrg.introghosts	txt.makescreen	
makescreen	- procedure -	scrn.add'info	txt.names	
pressure	----------	scrn.makehelp	txt.real time	

0.14 Data Base Disk

44 Files — 8 Blocks Free:

bootslow	- comal 0.14 -	filing	----------	- comal users -
fastboot	- programs -	free'form'db	dat.ffdb	- group, usa -
c64 comal 0.14	----------	guess'it	phone.dat	- 6041 monona -
comalerrors	1541 database	star'trek'db	ran.doctorwho	- madison, wi -
ml.sizzle	boot'data'base	telephone	ran.startrek	- 53716 -
hi	boot'tutor	tutor'amnesia	----------	----------
menu	data'base'mgr	tutor'remember	-for more info -	- 608-222-4432 -
window-rs232.obj	dbase14	----------	- send sase to -	----------
----------	doctorwhodb	- data files -	----------	

Holland Disk #9

117 Files — 16 Blocks Free:

```
hi                  edit-src            - data files -      pl0.test4           cod.test4
menu                show'asc            ----------------    ----------------    ----------------
----------------    ----------------    asc.on'irq          - xpl0 system -     - text files -
-comal programs-    -  external  -      asc.edit-src        -   files   -       ----------------
----------------    -  procedure -      scrn.edit-src       ----------------    txt.environment
demo/event0         ----------------    ----------------    - do not load -     txt.ffloat
demo/event1         ext.decompose       - xpl0 system -     ----------------    txt.grammar
demo/event2         ----------------    -  programs  -      xpl0data            txt.select input
demo/event3         -  function  -      ----------------    xpl0help            txt.xpl0
a cf grammar        ----------------    xpl0                xpl0mess            ----------------
envir/demo1         fnc.ffloat$         xpl0lexi            ----------------    - dutch comal -
envir/demo2         ----------------    xpl0comp            - xpl0 data -       - users group -
envir/demo3         - source code -     xpl0intp            -   files   -       ----------------
envir/demo4         ----------------    ----------------    ----------------    -dick klingens -
ffloat/demo         src.editdemo        - sample xpl0 -     - do not load -     -violier 3     -
haiku               src.environment     -  programs  -      ----------------    -krimpen ad    -
show'src            src.irq1.00         ----------------    lex.program         -       yssel-
test load           src.irq2.01         - use enter -       cod.program         -2925 te       -
test.extern         ----------------    -  to load  -       lex.test1           -the           -
tinycomal           -  packages  -      ----------------    cod.test1           -netherlands   -
demo/event4.0       ----------------    pl0.program         lex.test2           ----------------
demo/event4.1       pkg.environment     pl0.test1           cod.test2
demo/event4.2       pkg.irq             pl0.test2           lex.test3
demo/some events    ----------------    pl0.test3           lex.test4
```

England Disk #1

64 Files — 1 Blocks Free:

```
hi                  sort'u/l'case       ----- ----------    lst.dem'italics     obj.pllprt
----------------    prc.sort'anycase    prc.label'axis      -------- -------    es.pllprt'inst
  education         str$'no'display     prc.select'grad       music            ------------ ---
- --------------    -- -------------    lst.dem'heapsort    --------- ------      instructions
window'demo.2         engineering       clock utility       m'master.info       ------------- --
prc.setwindows      --- ------------    flpt->sci.not       m'master utility    read'about'disk
prc.window0         section             menu                playscore/2         about'disk
array'procs'demo    doc.section         ------ ---------    m/black & wh.rag    file'to'print
prc.print'timing    angle'data              fonts           m/bouree (bach)     letter
prc.random'array    perf                ------- --------    m/passion choral    -------------- -
prc.unsort'array    doc.perf            font editor 2.01    ---------- -----     v1.1  21/01/86
dem'str$'search2    ---- -----------    font editor         epson hires dump    ---------------
string$'search        utilities         font.slimfont       ----------- ----
```

England Disk #2

73 Files — 6 Blocks Free:

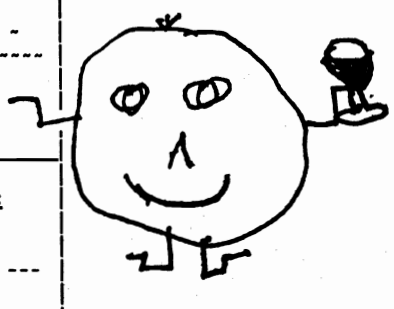
```
hi                  recursion-tree      ampersand           cycle'gear'data5    test'blas
----------------    -- -------------    skm.help            comalarmclock       sgefa
  education           utilities         &-fast              vt-52               src.blas
- --------------    --- ------------    &-no knocks         -------- -------    ------------ ---
triset'font.demo    bar'graph3          fast                    fonts             instructions
proc.type.3'sets    proc.trap'printr    no knocks           --------- ------    ------------- --
use'ext.f.dice      proc.blink'str$     ---- -----------        font.slim        read'about'disk
ext.func.dice       proc.yes'no+str$        music               font.triset      about'disk
proc.ext.handler    proc.italics'2      ----- ----------    ---------- -----     file'to'print
introd'quicksort    italics'window'2    coomposer               package          letter
demo'quicksort.2    screen'all'chars    variations          ---------- ----     -------------- -
proc.quicksort      proc.quote'keys     norweg.dance        pkg.blas             v1.0  16/03/86
proc.select'sort    peeks'to'obj        ------ ---------    file'to'view         ---------------
proc.bubble'sort    proc.802grafdump      applications      txt.about'blas
fivetrees           proc.fillscreen     ------- --------    txt.doc'blas
```

England Disk #3

65 Files — 0 Blocks Free:

```
hi                  u'gr.aid'u(i)       ext.i'scales3       proc.u'graph        proc.multigraph
u'graph.demo        u'gr.no3'u(i)       ext.i'scales4       proc.init'u'gr.     proc.file'graph
u'gr.aid'u'()       u'gr.expres'u(i)    ext.i'scales5       proc.u'window       proc.read'graph
u'gr.exam'u'()      u'gr.sin1'u()       ext.i'scalestxt     proc.xxx'u'data     hysteresis.demo
u'gr.brains'u'()    u'gr.sin2'u()       ext.opening'txt     proc.file'u'gr.p    - --------------
u'gr.sports'u'()    u'gr.sin1&2'u()     ext.i'grids         proc.read'u'gr.i      instructions
u'gr.paraf'u'()     u'gr.ran%'u()       ext.i'multi         ----------------    -- -------------
u'gr.xy'u'()        u'gr.exam'nos'()    ext.i'draw'on.1     proc.graph          read'about'disk
u'gr.random'u'()    ext.outline         ext.i'draw'on.2     demo'graph.pt.1     about'disk
u'gr.no3'u'()       ext.i'outline       ext.i'bars          demo'graph.pt.2     file'to'print
u'gr.locust'u'()    ext.i'types         ext.finale          proc.graph'init     letter
u'gr.neg'u'()       ext.i'scales1       ext.i'misc          proc.xxx'array      --- ------------
u'gr.expres'u'()    ext.i'scales2       ext.extras          proc.graphwindow    v1.1  30/11/86
```

TOASTTO COMAL 18



By Rhianon Lindsay

Specials

CP/M COMAL SPECIAL

We are adding support for what should prove to be a popular new COMAL implementation ... CP/M COMAL 2.10. It can run on the C128 in its CP/M mode, so many of our current readers can try it. It's significance is more than just another COMAL implementation though. It is the first COMAL here to include a "compiler". This RUNTIME compiler can turn a COMAL program into a stand alone program file. The program then will run without the need of COMAL. This RUNTIME compiler system usually costs $20 which is quite reasonable. RUNTIME systems for other languages often cost several hundred dollars or more. But to celebrate this first COMAL RUNTIME system, we are offering it to subscribers only for **just $4.95** if you get it at the same time as CP/M COMAL.

SCHOOL SPECIAL

It is a shame that more schools aren't teaching COMAL yet. COMAL is an ideal language for the schools. But many schools don't know about COMAL. If the school in your area has Commodore, IBM, or CP/M computers, why not make them aware of COMAL. For **just $10** we will provide a special **School COMAL Demo Pak**. It includes everything the school needs to preview COMAL:

Commodore 64 COMAL 0.14 Demo Disk
CP/M COMAL 2.10 Demo Disk
Mytech IBM PC COMAL 2.0 Demo Disk
Full set of *COMAL Today* issues 6-18
The 56 page *Index* to *COMAL Today*
Several copies of the INFO booklet

Send $10 to sponsor your local school. Include the name of the proper **person** at the school, along with the **school address**, and we will send this package to them. Or, we can send it directly to you ... and you can personally show them the wonders of COMAL.

TEXTBOOKS

Here is a chance to get a full sized COMAL textbook:

Beginning COMAL
Foundations in Computer Studies with COMAL
Introduction to Computer Programming with COMAL 2.0
COMAL Handbook

Take your pick, **only $9 each** (plus the usual $3 shipping).

SUPER CHIP

Super Chip has been so well received, that it now standard equipment! We are including a free Super Chip with every **C64 COMAL 2.0 Cartridge Deluxe Pak**. If you don't have a Super Chip in your black COMAL 2.0 cartridge, get one now for **just $10**. **Add $5** and we will include **Super Chip On Disk**.

MORE ...

We'd like to clear out some of our boxes on the shelves to make room for the new books coming out now. These specials are subject to a **$10 minimum order** (combine these with any order, even specials mentioned above).

Take your pick... **only $1 each** (plus shipping):

COMAL Today - The Index ($1 shipping)
COMAL From A to Z ($1 shipping)
COMAL Workbook ($1 shipping
Cheatsheet C64 Keyboard Overlay (folded-$1 shipping)

Specials Expire October 30, 1987

These specials are for **current** *COMAL Today* subscribers only. If you are a new subscriber, you can take advantage of these specials when you subscribe. ■

Easy Order Form

To order, circle the price. The lower price is COMAL Today subscribers only.

COMAL TODAY SUBSCRIPTION

$18.95 for 6 issues (start at current)

80 pages of interesting COMAL information and tips plus many complete program listings for you to type in and RUN.

TODAY DISK SUBSCRIPTION

$35.95 for 6 disks (start at current)

Companion to *COMAL Today*, each disk contains all the applicable programs for the Commodore 64. It is a double sided disk, with COMAL 0.14 programs on the front side, and COMAL 2.0 programs on the back side.

C64 COMAL 0.14 Starters Kit

$27.95 / $29.95

Includes everything you need to start programming in COMAL on the C64:

Two books:
- **COMAL From A to Z**
- **COMAL Workbook**

Four disks:
- **Tutorial Disk**
- **Auto Run Demo Disk**
- **Paradise Disk**
- **New Sampler Disk**

Six issues of *COMAL Today*
COMAL Today - The Index
Pocket Quick Guide Reference Card
COMAL INFO booklet

C64 COMAL 2.0 Deluxe Cart Pak

$128.90 / $138.95

For those who want only the best. Includes:

COMAL 2.0 Cartridge
Super Chip installed and tested
Four cartridge demo disks
Two books: (titles subject to change)
- **COMAL Handbook**
- **Cartridge Graphics and Sound**

CP/M COMAL 2.10

$49.95 / $69.95

For Commodore 128 in CP/M mode and any other CP/M computer. This includes a full featured COMAL 2.0 system packaged in a **Doc Box** with a manual and **Demo Disk**.

CP/M COMAL RUNTIME

$4.95 special / $39.95

This is an add on option to **CP/M COMAL 2.10** (above). It allows you to "compile" a COMAL program, creating a stand alone program that can be run directly from CP/M without the need of COMAL.

C64 COMAL 2.0 European Disks

$24.95 / $29.95

A dozen disks of COMAL programs and articles from Holland and England. The text and programs are generally in English. These disks include many programs or systems of programs that were too large for our normal *Today Disks*.

INTRODUCTION TO COMAL 2.0

$9.00 special / $19.95

A good tutorial book for C64 COMAL 2.0 cartridge users, written by a teacher and tested in the classroom. Spiral bound, it includes short "tests" at the end of each chapter and comes with a separate answer book.

COMAL HANDBOOK

$9.00 special / $18.95

470 page detailed reference to both C64 COMAL 0.14 and 2.0. Each keyword has its own pages, complete with syntax, examples, and full sample program showing it in use. Keywords are cross-referenced.

ORDER FORM Subscriber #_____ Name:_____

(more on other side)

July 1987 - Prices subject to change Street:_____

(only subscribers get lower prices)

Prepaid orders only, US Dollars, or charge: City/St/Zip:_____

VISA/MC #:_____exp date:_____Signature:_____

Circle price of items wanted: (first is DocBox pages price, second-subscriber price, third-list price - DocBox is pages only)

BOOKS - add $3 shipping per book (overseas extra) (Canada add $1 extra shipping per book)

DocBox	Sub price	List price	Item description - disks are Commodore format - DocBox is pages only
	$9.95	$14.95	New-Doc Box, mini 3 ring binder & slip case, cloth bound, quality D-ring style One Doc Box is required to hold books you buy as Doc Box pages!!! (see special below)
$15.95	-.-	$20.95	New-COMAL Cross Reference, Len Lindsay, *estimate over 200 pages* (due Aug 1987) *General* ‖ Detailed reference book for COMAL 2.0 implementations in USA (CP/M, Mytech, UniComal)
$12.95	-.-	$19.95	COMAL 2.0 Packages, Jesse Knight, *108 pages with disk* *C64 2.0 Advanced* ‖ How to write a package in Machine Code; includes C64 comsymb & supermon
$12.95	$17.95	$19.95	Packages Library Volume 1, David Stidolph, *76 pages with disk* *C64 2.0* ‖ 17 example packages ready to use, many with source code, plus the Smooth Scroll Editor
$12.95	-.-	$19.95	New-Packages Library Volume 2, *estimate 50 pages with disk* (due Aug 1987) *C64 2.0* ‖ more example packages ready to use, many with source code
$9.95	$14.95	$15.95	Graph Paper, Garrett Hughes, *52 pages with disk* (now in stock) *C64 2.0* ‖ Function graphing system for C64 COMAL 2.0. The program can't be LISTed.
$15.95	-.-	$20.95	New-COMAL Collage, Frank & Melody Tymon, *168 pages with disk* (now in stock) *C64 2.0* ‖ Graphics and Sprites tutorial with many full sized example programs
$12.95	-.-	$18.95	New- 3 Programs in Detail, Doug Bittenger, *book and disk* (due Aug 1987) *C64 2.0* ‖ A step by step guide to creating: Blackbook, Home Accountant, BBS
$6.95	-.-	$11.95	New-Today Tutorials, From COMAL Today #1-17, updated as needed (due Aug 1987) *General* ‖ Includes explanations of structures, procedures, functions, and string handling
$6.95	-.-	$11.95	New-Today Tips & Notes, Collection of programming tips and notes (due Aug 1987) *C64 0.14 & 2.0* ‖ Taken from COMAL Today issues #1-17, updated as needed
$4.95	$4.95	$6.95	Cartridge Graphics & Sound, Captain COMALs Friends, *64 pages* *C64 2.0* ‖ A reference book to the 11 packages built into the C64 COMAL 2.0 cartridge
$17.95	$22.95	$24.95	Cartridge Tutorial Binder, Frank Bason & Leo Hojsholt, *320 pages with disk* *C64 2.0* ‖ Now available punched for our Doc Box! The tutorial from Commodore Denmark
-.-	$4.95	$6.95	COMAL Today - The Index, Kevin Quiggle, *52 pages with disk,* *General* ‖ 4,848 entry index to COMAL Today issues 1-12, full data disk files & reader program
$9.95	-.-	$14.95	3rd printing-Library of Functions & Procedures, Kevin Quiggle, *80 pages with disk* *C64 0.14* ‖ Reprinted by popular demand. Over 140 procedures & functions ready to use
-.-	$17.95	$19.95	Introduction to Computer Programming with COMAL, J William Leary, *272 pages* *C64 2.0* ‖ Beginners text book, spiral bound, now includes separate answer book
-.-	$16.95	$18.95	COMAL Handbook, Len Lindsay, *479 pages* *C64 0.14 & C64 2.0* ‖ Detailed reference book for C64 COMAL 0.14 and 2.0
-.-	$17.95	$19.95	Foundations in Computer Studies With COMAL, John Kelly, *363 pages* *General* ‖ Beginners text book, Jr/Sr High School level
-.-	$18.95	$20.95	Beginning COMAL, Borge Christensen, *333 pages* (imported from England) *General* ‖ Beginners text book, Elementary School level, written by the founder of COMAL
-.-	$4.95	$6.95	COMAL From A to Z, Borge Christensen, *64 pages* *C64 0.14* ‖ Mini reference guide to C64 COMAL 0.14 by the founder of COMAL
-.-	$12.95	$14.95	Captain COMAL Gets Organized, Len Lindsay, *102 pages with disk* *C64 0.14* ‖ Application tutorial to illustrate benefits of modular programming
-.-	$4.95	$6.95	COMAL Workbook, Gordon Shigley, *69 pages* *C64 0.14* ‖ Companion to the Tutorial Disk, great for beginners, full sized fill in the blank style
-.-	$12.95	$14.95	Graphics Primer, Mindy Skelton, *84 pages with disk* (only a couple left) *C64 0.14* ‖ Beginners graphics and sprites tutorial
Special:		$0.00	Free Doc Box with over $20 of Doc Box style pages (expires 9/30/87, one per subscriber)
	$5.00		Doc Box only $5 with any one Doc Box pages book (offer expires 9/30/87)

--->» Add the total for items on this side to the bottom of the other side. Shipping add $3 per book.

Then mail to: COMAL Users Group, USA, Limited, 6041 Monona Drive, Madison, WI 53716

ORDER FORM Subscriber #_____ Name:_____

(more on other side)

July 1987 - Prices subject to change Street:_____

(only subscribers get lower prices)

Prepaid orders only, US Dollars, or charge: City/St/Zip:_____

VISA/MC #:_____ exp date:_____ Signature:_____

To order, circle price or check box as indicated:

NEWSLETTER & DISK SUBSCRIPTIONS (and single issues)

$18.95 (6 issues) **COMAL Today** newsletter subscription--> [___]renew [___]start at current issue
$26.95 (10 issues) (Canada add $1 per issue; Overseas add $5 per issue)
$3.95 each Backissues COMAL Today->circle issues wanted: 5 6 7 8 9 10 11 12 13 14 15 16 17
$14.95 (subscribers only) COMAL Yesterday, first 4 issues of COMAL Today, spiral bound
$35.95 (6 disks) **Today Disk** subscription--> [___]renew [___]start at current [___]start at #_____
$55.95 (10 disks)
$9.75 each Today Disk-> circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

SYSTEMS:

Sub price	List price	Item Description - disks are Commodore format unless otherwise stated
$4.00	$5.00	**New-CP/M COMAL 2.10 Final Demo disk** (the SAVE and ENTER commands are disabled)
$49.95	$69.95	**New-CP/M COMAL 2.10 Final Full System, manual & disk in Doc Box** (ship $5)
$19.95	$39.95	**New-CP/M COMAL RUNTIME System (creates stand alone programs like a compiler)** (CP/M COMAL & RUNTIME run on the C128 in CP/M mode ... disk is in CP/M format)
$27.95	$29.95	C64 COMAL 0.14 Starters Kit (*4 disks, 2 books, 6 newsletters, more*)-(ship $4)
$128.90	$138.95	C64 COMAL 2.0 Cartridge Deluxe Pak (*cart, superchip, 2 books, 4 disks*) (ship $5)
$89.95	$99.95	C64 COMAL 2.0 Cartridge plain (*no manual, no disks*) (ship $2)
$24.95	$29.95	Super Chip for black C64 COMAL 2.0 Cartridge (*add $5 for installation/testing fee*) (ship $1)
$185.00	$199.95	**New-C128 COMAL 2.0 Cartridge** (*cartridge, manual, demo disk*) (special order only) (ship $5)
$9.75	$14.95	Commodore PET 32K computer COMAL 0.14 (*disk only*)
$595.00	$595.00	UniComal IBM PC COMAL 2.1 (*English manual*) (special order only) (ship $5)
$4.00	$5.00	**New-Mytech preliminary IBM PC COMAL 2.0 Demo disk** (full system not available yet)
-.-	-.-	*Coming soon ... Apple COMAL and Final Mytech IBM PC COMAL* (under $100 each)

DISK SETS:

$24.95 $29.95 **New-Full set of 12 European 2.0 Program disks** (3 from England & 9 from Holland) (ship $2)
$14.95 $24.95 All four C64 2.0 Cart Demo Disks (1,2,3,4) (ship $2)
$24.95 $29.95 **Full set of eleven 0.14 User Group disks** (1,2,3,4,5,6,7,8,9,10,12) (ship $2)
$14.95 $19.95 **Full set of four 2.0 User Group disks** (11,13,14,15) (ship $2)
$9.75 $10.00 Single User Group Disk; circle which one--> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
$19.95 $24.95 **Super Chip On Disk plus Super Chip Programs Disk**
$49.95 $99.95 **New-Source Code to Super Chip** (requires Commodore's Assembler & Packages Library Fixes)

SPECIALTY DISKS ($9.75 each) (check box to order)

[___]**DataBases**(0.14/2.0) [___]TutorialDisk(0.14) [___]Utility #1(0.14) [___]**SuperChip Programs**
[___]Font(0.14/2.0) [___]AutoRunDemo(0.14) [___]Utility #2(0.14) [___]Shareware2.0(3 sides)
[___]Games(0.14/2.0) [___]ParadiseDisk(0.14) [___]SlideShow#1&2(0.14) [___]Read & Run(2.0)
[___]Modem(0.14/2.0) [___]BricksTutorials(0.14) [___]SpanishCOMAL(0.14) [___]Math & Science(2.0)
[___]**New-Full 0.14 Sampler** from San Francisco show [___]Articles#1(textfiles) [___]Typing(2.0)

Total $_____ + $_____ shipping (this side) -- **Shipping is $3 per book**
Total $_____ + $_____ shipping (other side) -- **Minimum shipping charge of $2 per order**

Total $_____ + $_____ = US$_____ **Total Paid** (WI add 5% sales tax) (overseas is extra)

Mail to: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call (608) 222-4432

(allow two weeks for checks to clear ... $10 charge for checks not honored by the bank)

HOW TO DO IT IN COMAL

How to see what is on a disk:

Put the disk into the drive. Type:
cat

How to retrieve a program from disk:

Put the disk into the drive. Type:
load "name"

How to run a program that is in the computer:

Type the command:
run

How to run a program from disk:

Put the disk into the drive. Type:
run "name"

How to look at a program:

If the program is not in the computer, load it first. To see the program in the computer type:
list

How to write a new program:

Type the commands:
new
auto (now type in the program)

How to add lines to the end of a program:

The program must be in the computer. Type:
auto (now type in the lines)

How to add lines in the middle of a program:

The program must be in the computer. Locate the place you wish to add a line. For example:
0040 print "welcome to my program"
0050 print "hope you enjoy it"

Use a line number between the lines. Type:
45 print "written just for you"

How to erase lines in a program:

the line number of the line to be erased (50 in this example). Type:
del 50

You may delete a block of lines all at once. Find the first and last line numbers in the block of lines (lines 500 through 750 for this example), then issue the DEL command:
del 500-750

You also may delete a procedure or function by name. Type:
del name

How to renumber a program:

To restore lines numbered by 10's type:
renum

How to store a program on disk:

Type the command:
save "name"

How to list a program on the printer:

The program must be in the computer. Type:
list "lp:" (most COMALs use "lp:" for printer)

How to store program lines as ASCII file:

Example, to store procedure **pause**, type:
list pause "pause.lst" «--most COMALs
list "pause.lst" pause «--CP/M COMAL

How to merge procedures into other programs:

Example, to merge **pause** into current program:
merge "pause.lst"

How to exit COMAL:

To return to the main operating system, type:
bye «--most COMALs
basic «--C64 & C128

This page of COMAL information has been brought to you by the good people at:

The Transactor

501 Alden Road, PO Box 3250, Markham Industrial Park, Markham, Ontario L3R 9Z9 Canada

Common COMAL Keywords

keyword	c	8	i	p	m
// (comment)	■	■	■	■	■
abs	■	■	■	■	■
and	■	■	■	■	■
append	■	■	■	■	■
at	■	■	-	■	■
atn	■	■	■	■	■
auto	■	■	■	■	■
basic / bye	■	■	■	■	■
bitand	■	■	■	■	▩
bitor	■	■	■	■	▩
bitxor	■	■	■	■	▩
bye / basic	■	■	■	■	■
case	■	■	■	■	■
cat / dir	■	■	■	■	■
chain	■	■	■	■	■
change	■	■	■	■	-
chr$	■	■	■	■	■
close	■	■	■	■	■
closed	■	■	■	■	■
con	■	■	■	■	■
cos	■	■	■	■	■
create	■	■	■	■	■
curcol	▪	▪	■	■	-
currow	▪	▪	■	■	-
cursor	■	■	■	■	■
data	■	■	■	■	■
del	■	■	■	■	■
delete	■	■	■	■	■
dim	■	■	■	■	■
dir / cat	■	■	■	■	■
discard	■	■	■	■	■
display	■	■	■	▪	▪
div	■	■	■	■	■
do	■	■	■	■	■
edit	■	■	■	■	■
elif	■	■	■	■	■
else	■	■	■	■	■
end	■	■	■	■	■
endcase	■	■	■	■	■
endfor / next	■	■	■	■	■
endfunc	■	■	■	■	■
endif	■	■	■	■	■
endloop	■	■	■	■	■
endproc	■	■	■	■	■
endtrap	■	■	■	■	■
endwhile	■	■	■	■	■
enter	■	■	■	■	■
eod	■	■	■	■	■
eof	■	■	■	■	■

keyword	c	8	i	p	m
err	■	■	■	■	■
errtext$	■	■	■	■	■
esc	■	■	■	■	■
exec	■	■	■	■	■
exit	■	■	■	■	■
exp	■	■	■	■	■
external	■	■	■	■	■
false	■	■	■	■	■
file	■	■	■	■	■
find	■	■	■	■	■
for	■	■	■	■	■
free	▪	▪	■	■	-
func	■	■	■	■	■
get$	■	■	■	■	■
goto	■	■	■	■	■
handler	■	■	■	■	■
if	■	■	■	■	■
import	■	■	■	■	■
in	■	■	■	■	■
input	■	■	■	■	■
int	■	■	■	■	■
key$	■	■	■	■	■
label	■	■	■	■	▪
len	■	■	■	■	■
let (assign)	■	■	■	■	■
link	■	■	■	■	■
list	■	■	■	▪	▪
load	■	■	■	■	■
log	■	■	■	■	■
loop	■	■	■	■	■
main	■	■	■	■	-
merge	■	■	■	■	■
mod	■	■	■	■	■
mount	■	■	■	■	■
new	■	■	■	■	■
next / endfor	■	■	■	■	■
not	■	■	■	■	■
null	■	■	■	■	■
of	■	■	■	■	■
open	■	■	■	■	■
or	■	■	■	■	■
ord	■	■	■	■	■
otherwise	■	■	■	■	■
page	■	■	■	■	■
peek	■	■	▪	■	-
pi	■	■	■	■	■
poke	■	■	▪	■	-
print	■	■	■	■	■
proc	■	■	■	■	■

keyword	c	8	i	p	m
random	■	■	■	■	■
randomize	■	■	■	■	■
read	■	■	■	■	■
ref	■	■	■	■	■
rename	■	■	■	■	■
renum	■	■	■	■	■
repeat	■	■	■	■	■
report	■	■	■	■	■
restore	■	■	■	■	■
return	■	■	■	■	■
rnd	■	■	■	■	▪
run	■	■	■	■	■
save	■	■	■	■	■
scan	■	■	■	■	■
select output	■	■	▪	■	▪
sgn	■	■	■	■	■
sin	■	■	■	■	■
size	■	■	■	■	■
spc$	■	■	■	■	■
sqr	■	■	■	■	■
step	■	■	■	■	■
stop	■	■	■	■	■
str$	■	■	■	■	■
tab	■	■	■	▪	■
tan	■	■	■	■	■
then	■	■	■	■	■
to	■	■	■	■	■
trap	■	■	■	■	■
true	■	■	■	■	■
unit$ (chdir)	■	■	■	■	▪
until	■	■	■	■	■
use	■	■	■	▪	■
using	■	■	■	■	■
val	■	■	■	■	■
when	■	■	■	■	■
while	■	■	■	■	■
write	■	■	■	■	■
zone	■	■	■	■	▪

c = c64 COMAL 2.01 cartridge
8 = c128 COMAL 2.02 cart
i = IBM MS-DOS Unicomal 2.1
p = CP/M COMAL 2.10
m = IBM PC Mytech 2.0

■ = fully implemented
▪ = package/slight syntax dif
▮ = planned for final
- = not implemented